INFORUM 2015

Atas do 7.º Simpósio Nacional de Informática

7 e 8 de Setembro de 2015
Universidade da Beira Interior
Portugal

Editores

Luís Veiga, Instituto Superior Técnico
Ricardo Rocha, Universidade do Porto

Ficha Técnica

Título. **INFORUM 2015 - Atas do 7.º Simpósio Nacional de Informática**
Autores. **Luís Veiga, Ricardo Rocha**
ISBN. **978-989-654-248-1**
Ano. **2015**
Editora. **UBI - Universidade da Beira Interior. Serviços Gráficos**

Publicado em formato eletrónico no seguinte endereço web:
http://inforum.org.pt/INForum2015/docs/atas-do-inforum2015

Organização

Apoios

Prefácio

Estas atas correspondem à 7ª edição do Simpósio Nacional de Informática, INForum 2015, que se realizou na Universidade da Beira Interior, nos dias 7 e 8 de setembro de 2015.

O INForum é um evento anual de reunião da comunidade nacional nas diversas vertentes da Informática, constituindo um fórum privilegiado para a apresentação, divulgação e discussão de trabalhos científicos e avanços tecnológicos em diferentes áreas da Informática. Nesta edição do INForum realizam-se sete sessões temáticas:

- Ciência e Engenharia de Software
- Computação e Conteúdos Culturais
- Computação Móvel e Ubíqua
- Computação Paralela, Distribuída e de Larga Escala
- Gestão de Dados e Conhecimento
- Segurança de Sistemas de Computadores e Comunicações
- Sistemas Embebidos e de Tempo-Real

Nesta edição foram aceites dois tipos de contribuições: (i) artigos, apresentando resultados de trabalho de investigação científica realizado em contexto académico ou industrial, e (ii) comunicações, cujo objetivo é permitir a divulgação de uma forma mais expedita e informal de trabalho de I&D desenvolvido na comunidade. No total, o conjunto de sessões atraiu a submissão de 77 trabalhos, 60 artigos e 17 comunicações.

Cada artigo foi sujeito a um processo de revisão seletivo pela comissão de programa da sessão respetiva, tendo sido, na quase totalidade dos casos, avaliado por pelo menos três revisores. A aceitação de cada comunicação foi decidida igualmente pela comissão de programa da sessão respetiva, com base na análise dos resumos estendidos submetidos. No final, foram selecionados 30 artigos, que estão incluídos nestas atas, e 15 comunicações. Todos os trabalhos aceites, artigos ou comunicações, poderam ainda contribuir adicionalmente com a exposição de um poster.

O programa do evento incluiu ainda duas palestras convidadas e uma sessão com representantes da Indústria. As palestras convidadas foram proferidas por Yves Bertot, do INRIA, e José Alegria, da PT Portugal. Os resumos de ambas as palestras estão incluídos nestas atas.

O trabalho da Comissão de Programa apenas foi possível devido à preciosa colaboração que recebemos de diversos colegas e entidades. À Comissão Coordenadora, agradecemos a confiança depositada e ao seu presidente, António Casimiro, agradecemos a disponibilidade, orientação e apoio prestado. Aos coordenadores das várias sessões e respetivas comissões de programa, agradecemos todo o esforço e cuidado colocado na análise e discussão dos trabalhos submetidos. Aos oradores convidados, agradecemos a gentileza de terem aceite enriquecer o programa do INForum com a sua presença. Aos membros do júri para a atribuição do prémio de melhor artigo, um agradecimento especial por terem aceite

o convite e pela colaboração prestada. À Comissão Organizadora, coordenada pelo Simão Sousa, agradecemos todo o apoio prestado e o excelente trabalho de organização.

Estamos igualmente reconhecidos à Universidade da Beira Interior, pelas excelentes condições diponibilizadas para acolher o evento, aos autores do software de gestão de conferências EasyChair, pela inestimável ajuda que nos proporcionaram, e a todos os patrocinadores, parceiros e apoiantes do INForum.

Por último, o nosso especial obrigado vai para todos os autores dos trabalhos submetidos, por manterem viva a chama do INForum, e para todos os participantes, por fazerem do INForum um evento privilegiado de promoção e intercâmbio de conhecimento da comunidade nacional de Informática.

Agosto 2015

Luís Veiga
Ricardo Rocha

INFORUM 2015 - Organização

Presidência da Comissão de Programa

Luís Veiga	Instituto Superior Técnico
Ricardo Rocha	Universidade do Porto

Comissão Organizadora

Coordenação

Simão Melo de Sousa	Universidade da Beira Interior

João Paulo Cordeiro	Universidade da Beira Interior
João Paulo Fernandes	Universidade da Beira Interior
José Luís Faria	Universidade do Minho (webmaster)
Paul Crocker	Universidade da Beira Interior
Paulo Fazendeiro	Universidade da Beira Interior

Comissão Coordenadora

Coordenação

António Casimiro	Universidade de Lisboa

Ademar Aguiar	Universidade do Porto
Antónia Lopes	Universidade de Lisboa
Beatriz Sousa Santos	Universidade de Aveiro
João Pascoal Faria	Universidade do Porto
José Orlando Pereira	Universidade do Minho
Luís Caires	Universidade Nova de Lisboa
Luís S. Barbosa	Universidade do Minho
Luís Veiga	Instituto Superior Técnico
Miguel P. Correia	Instituto Superior Técnico
Raúl Barbosa	Universidade de Coimbra
Ricardo Rocha	Universidade do Porto
Salvador Abreu	Universidade de Évora
Simão Melo de Sousa	Universidade da Beira Interior
Teresa Gonçalves	Universidade de Évora

Comissões de Programa das Sessões Temáticas

Ciência e Engenharia de Software (SOFT-PT)

Coordenação

Vasco Thudichum Vasconcelos	Universidade de Lisboa
António Rito Silva	Instituto Superior Técnico
Bruno Cabral	Universidade de Coimbra
Carla Ferreira	Universidade Nova de Lisboa
Dimitris Mostrous	Universidade de Lisboa
Eduardo Marques	Universidade de Lisboa
Gonçalo Oliveira	Frotcom International
Hugo Lourenço	OutSystems
João Pascoal Faria	Universidade do Porto
João Pina	Novabase
Jorge Sousa Pinto	Universidade do Minho
José Luís Oliveira	Universidade de Aveiro
Luís Barbosa	Universidade do Minho
Luís Caires	Universidade Nova de Lisboa
Luís Lopes	Universidade do Porto
Marco Vieira	Universidade de Coimbra
Mário Florido	Universidade do Porto
Nuno Silva	Critical Software SA
Pedro Adão	Instituto Superior Técnico
Salvador Abreu	Universidade de Évora
Simão Melo de Sousa	Universidade da Beira Interior

Computação e Conteúdos Culturais (CCC)

Coordenação

António Coelho	Universidade do Porto
Cristina Sá	Universidade Católica Portuguesa
Matthew Davies	INESC TEC
Paula Viana	Instituto Superior de Engenharia do Porto
Carlos Vaz de Carvalho	Instituto Superior de Engenharia do Porto
Fabien Gouyon	Universidade do Porto
Gilberto Bernardes	Universidade do Porto
Heitor Alvelos	Universidade do Porto
Leonel Morgado	Universidade Aberta
Maximino Bessa	Universidade de Trás-os-Montes e Alto Douro
Miguel Carvalhais	Universidade do Porto
Miguel Leitão	Instituto Superior de Engenharia do Porto
Nelson Zagalo	Universidade do Minho

Nuno Correia	Universidade Nova de Lisboa
Paula Escudeiro	Instituto Superior de Engenharia do Porto
Rui Nóbrega	Universidade Nova de Lisboa
Rui Penha	Universidade do Porto
Rui Rodrigues	Universidade do Porto
Teresa Andrade	Universidade do Porto
Teresa Chambel	Universidade de Lisboa
Teresa Romão	Universidade Nova de Lisboa

Computação Móvel e Ubíqua (CMU)

Coordenação

Adriano Moreira	Universidade do Minho
Ana Aguiar	Universidade do Porto
Helena Rodrigues	Universidade do Minho

Ana Paula Afonso	Universidade de Lisboa
António Coelho	Universidade do Porto
Carlos Baquero	Universidade do Minho
Dulce Domingues	Universidade de Lisboa
Filipe Pacheco	Instituto Politécnico do Porto
Frutuoso Silva	Universidade da Beira Interior
Hugo Miranda	Universidade de Lisboa
Jorge Sá Silva	Universidade de Coimbra
José Maria Fernandes	Universidade de Aveiro
Nuno Santos	Instituto Superior Técnico
Paulo Ferreira	Instituto Superior Técnico
Pedro Brandão	Universidade do Porto
Ricardo Morla	Universidade do Porto
Rui Prior	Universidade do Porto
Sérgio Duarte	Universidade Nova de Lisboa
Susana Sargento	Universidade de Aveiro

Computação Paralela Distribuída e de Larga Escala (CPDLA)

Coordenação

João Barreto	Instituto Superior Técnico
João Leitão	Universidade Nova de Lisboa

Alysson Bessani	Universidade de Lisboa
Filipe Araújo	Universidade de Coimbra
Henrique Moniz	Universidade Nova de Lisboa
Hervé Paulino	Universidade Nova de Lisboa
Hugo Miranda	Universidade de Lisboa
João Lourenço	Universidade Nova de Lisboa

João Silva	Instituto Superior Técnico
José Pereira	Universidade do Minho
Lúcio Ferrão	OutSystems
Luís Rodrigues	Instituto Superior Técnico
Manuel Barata	Instituto Superior de Engenharia de Lisboa
Miguel Matos	Universidade do Minho
Nuno Preguiça	Universidade Nova de Lisboa
Paula Prata	Universidade da Beira Interior
Ricardo Dias	Universidade Nova de Lisboa
Ricardo Vilaça	Universidade do Minho
Vítor Duarte	Universidade Nova de Lisboa

Gestão de Dados e Conhecimento (GDC)

Coordenação

Helena Galhardas	Instituto Superior Técnico
Paulo Maio	Instituto Superior de Engenharia do Porto
Rui Camacho	Universidade do Porto

Alberto Simões	Instituto Politécnico do Porto
Bruno Martins	Instituto Superior Técnico
Carlos Ferreira	INESC Porto
Carlos Soares	Universidade do Porto
Cristina Ribeiro	Universidade do Porto
David Matos	Instituto Superior Técnico
Diana Santos	University of Oslo
Fernando Batista	ISCTE-IUL
Francisco Couto	Universidade de Lisboa
Hugo Gonçalo Oliveira	Universidade de Coimbra
Irene Rodrigues	Universidade de Évora
João Magalhães	Universidade Nova de Lisboa
João Dias Pereira	Instituto Superior Técnico
João Paulo Cordeiro	Universidade da Beira Interior
João Pedro Silva	TEBE, S.A.
Jorge Barbosa	Universidade do Porto
Luís Teixeira	Fraunhofer Portugal AICOS
Luísa Coheur	Instituto Superior Técnico
Nuno Mamede	Instituto Superior Técnico
Nuno Silva	Instituto Superior de Engenharia do Porto
Paulo Gomes	Universidade de Coimbra
Paulo Novais	Universidade do Minho
Paulo Oliveira	Instituto Superior de Engenharia do Porto
Paulo Quaresma	Universidade de Évora
Pável Calado	Instituto Superior Técnico
Ricardo Ribeiro	ISCTE-IUL

Segurança de Sistemas de Computadores e de Comunicações (SSSC)

Coordenação

Miguel Pardal	Instituto Superior Técnico
Miguel Pupo Correia	Instituto Superior Técnico
Carlos Ribeiro	Instituto Superior Técnico
Carlos Serrão	ISCTE-IUL
Dulce Domingos	Universidade de Lisboa
Edmundo Monteiro	Universidade de Coimbra
Henrique Domingos	Universidade Nova de Lisboa
Henrique Santos	Universidade do Minho
José Alegria	PT - Portugal Telecom
José Carlos Lourenço Martins	Academia Militar
José Manuel Valença	Universidade do Minho
Luís Antunes	Universidade do Porto
Manuel Barbosa	Universidade do Minho
Marco Vieira	Universidade de Coimbra
Mário M. Freire	Universidade da Beira Interior
Mário Zenha Rela	Universidade de Coimbra
Nuno Neves	Universidade de Lisboa
Nuno Santos	Instituto Superior Técnico
Paulo Nunes	Academia Militar
Paulo Sousa	Maxdata Software
Pedro Adão	Instituto Superior Técnico
Pedro Inácio	Universidade da Beira Interior
Ricardo Chaves	Instituto Superior Técnico
Rui Miguel Silva	Instituto Politécnico de Beja
Salvador Pinto Abreu	Universidade de Évora
Susana Sargento	Universidade de Aveiro

Sistemas Embebidos e de Tempo-Real (SETR)

Coordenação

José Rufino	Universidade de Lisboa
Luís Lino Ferreira	Instituto Superior de Engenharia do Porto
Luís Miguel Pinho	Instituto Superior de Engenharia do Porto
Aleksandar Ilic	Instituto Superior Técnico
Carlos Almeida	Instituto Superior Técnico
Hélder Silva	EDISOFT, SA.
João Cardoso	Universidade do Porto
João Cunha	Instituto Superior de Engenharia de Coimbra
Joaquim Ferreira	Universidade de Aveiro

Jorge Pinto	Universidade do Minho
José Faria	Educed Lda
José Neves	GMV
Luís Almeida	Universidade do Porto
Luís Gomes	Universidade Nova de Lisboa
Mário Calha	Universidade de Lisboa
Nelma Moreira	Universidade do Porto
Nelson Blanco	PDMFC
Paulo Bartolomeu	Universidade de Aveiro
Paulo Pedreiras	Universidade de Aveiro

Revisores Adicionais

Bernardo Ferreira
Bernardo Portela
Diogo Lima
Helder Oliveira
João A. Silva
José Pedro
Nuno Cruz
Óscar Pereira
Pedro Saleiro
Pedro Salgueiro
Rui Caldeira
Sérgio Crisóstomo
Valter Balegas

Índice

Ciência e Engenharia de Software (SOFT-PT)

Computação e Conteúdos Culturais (CCC)

Computação Móvel e Ubíqua (CMU)

Computação Paralela Distribuída e de Larga Escala (CPDLA)

Gestão de Dados e Conhecimento (GDC)

Segurança de Sistemas de Computadores e de Comunicações (SSSC)

Sistema de Suporte à Execução de uma Linguagem Web Reativa

João Mateus, Miguel Domingues and João Costa Seco

Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa

Resumo O desenvolvimento de *software* segundo metodologias ágeis é dividido num conjunto de etapas que são sucessivamente repetidas, onde se produzem estados intermédios dos sistemas a desenvolver. No caso das aplicações centradas em dados persistentes, como as aplicações *web* e *cloud*, o uso destas metodologias resulta na evolução contínua do código e respetivo esquema de dados. Devido às constantes modificações, a probabilidade de introdução de erros de programação aumenta, principalmente quando é necessário manter a consistência entre código e dados. A uma escala mais pequena, as tarefas de programação, compilação, teste e *deployment* também prejudicam a produtividade do programador, pois adiam para o fim do ciclo informação relevante sobre o funcionamento da aplicação.
Neste artigo propomos uma solução baseada numa linguagem de programação e num ambiente de desenvolvimento *live*, que fornece *feedback* constante e imediato quanto ao estado atual da aplicação (código e dados). O sistema garante, estaticamente, que todas as modificações efetuadas à aplicação são isentas de erros. O sistema suporta ainda, concorrentemente, a evolução e execução de aplicações centradas em dados persistentes.

Keywords: Programação *live*, evolução de *software*, linguagem reativa de *data-flow*, aplicações *web*

1 Introdução

O desenvolvimento de aplicações segundo metodologias ágeis segue um ciclo de desenvolvimento composto por diversas etapas (escrita de código, compilação, teste, *deployment*). Este ciclo é iterado múltiplas vezes, adicionando incrementalmente funcionalidades à aplicação. Em particular, no caso de aplicações *web*, o ciclo de desenvolvimento divide-se em: especificação, implementação, compilação, teste e *deployment*. Só depois de concluída a primeira iteração, e respetivo primeiro *deployment*, é possível utilizar as funcionalidades especificadas. As diversas etapas desde a implementação da aplicação até à sua execução no servidor impedem o programador de ter, de modo imediato, uma noção clara de como a aplicação funciona e se cumpre a especificação inicial. O processo de compilação e *deployment* distanciam o programador da execução da aplicação, o que pode prejudicar a sua produtividade, ao atrasar *feedback* potencialmente valioso sobre

o comportamento da aplicação. Além disso, durante o processo de evolução de aplicações centradas em dados, também pode haver a introdução de novos *bugs*, derivados da evolução colateral do código e do respetivo esquema de dados [8]. A utilização de uma ferramenta de programação que fornece *feedback* imediato ajuda a diminuir a distância entre a implementação e a execução, permitindo ao programador implementar e utilizar a aplicação em simultâneo. Neste artigo, apresenta-se um ambiente de desenvolvimento *live* que permite visualizar o efeito que cada modificação tem no funcionamento da aplicação, aumentando assim a produtividade e eficiência do processo de desenvolvimento [3,10]. Para além do *feedback*, a reconfiguração incremental do sistema sem a necessidade de o interromper ou reiniciar tem a vantagem de evitar quebras de serviço, mantendo as aplicações sempre disponíveis.

A nossa abordagem consiste na extensão e implementação de uma linguagem incremental e reativa, que permite desenvolver aplicações *web* centradas em dados persistentes de um modo incremental e mantendo o estado da aplicação sempre atualizado [2]. Cada um dos incrementos é verificado estaticamente pelo sistema de tipos da linguagem, garantindo que a aplicação evolui de forma segura. Após a verificação estática de cada incremento, este é introduzido na aplicação em execução, sendo que o *deployment* é feito de forma transparente e imediata, sem qualquer interrupção ou disrupção do sistema. Desta forma, elimina-se a necessidade de duas etapas distintas para compilação e *deployment* [1].

Para fornecer *feedback* imediato, a aplicação precisa de refletir os dados e o código mais recentes. Uma solução passa por conceder propriedades reativas à aplicação, que garantem a propagação de todas as alterações efetuadas. A reatividade suportada pela linguagem tem por base um grafo de dependências entre os vários elementos da aplicação. Sempre que um destes elementos é modificado, essa alteração é propagada pelo grafo, atualizando todos os elementos cujas dependências tenham sofrido alterações. Desta forma, o sistema garante que todos os elementos estão sincronizados entre si e que a aplicação reflete o código e dados que estão presentes no sistema. De modo a evitar que a propagação das alterações entre num ciclo infinito, é necessário que o grafo de dependências seja acíclico, algo que é garantido estaticamente pelo sistema de tipos. Ao combinar o desenvolvimento incremental com a reatividade, conseguimos fornecer ao programador um ambiente de desenvolvimento *live*, onde todas as alterações efetuadas ao sistema podem ser vistas e utilizadas em tempo-real na aplicação em execução, melhorando a experiência de desenvolvimento do programador.

No contexto deste projeto, foi desenvolvido um protótipo [9] onde se representam os conceitos anteriores. O sistema presente neste protótipo fornece um ambiente de desenvolvimento (IDE), acessível através do *browser*, onde se pode programar o comportamento e a interface de uma aplicação *web*, combinado na mesma linguagem o *front-end* (interface HTML) e o *back-end* (lógica e esquema de dados). A semântica da linguagem é mapeada para elementos de uma aplicação *web*. Por exemplo, uma função que devolve um documento HTML é convertida para uma página HTML com parâmetros. Para propagar as alterações pelo grafo de dependências, o sistema utiliza uma fila de operações onde são adicionados os

nomes dos elementos que necessitam de ser atualizados. Os clientes (*browsers*) utilizam o padrão do observador em relação aos elementos de uma página. Sempre que o valor associado a um elemento muda, é enviada uma mensagem via WebSockets para as páginas (clientes) que estão, nesse momento, subscritas a esse nome, indicando qual o novo valor.

As principais contribuições deste projeto são:

- Uma linguagem de programação, orientada ao desenvolvimento de aplicações *web* reativas, permitindo implementar a lógica, esquema de dados e interface;
- Um sistema que possibilita o desenvolvimento de aplicações *web* de modo incremental, garantindo estaticamente que a evolução do sistema não causa qualquer disrupção;
- Um ambiente de desenvolvimento *live* que oferece ao programador *feedback* imediato e contínuo quanto ao seu código, conferindo propriedades reativas às aplicações que mantêm a interface sincronizada com o código e dados.

Na Secção 2 apresenta-se detalhadamente a linguagem através de um exemplo construido incrementalmente. De seguida, a Secção 3 descreve o sistema de suporte à execução, e as Secções 4 e 5 detalham o trabalho relacionado e trabalho futuro, respetivamente. Por fim, na Secção 6 é feita uma discussão final do projeto.

2 Linguagem

O foco da linguagem aqui apresentada é possibilitar o desenvolvimento dos aspetos base de uma aplicação *web* reativa centrada em dados persistentes, i.e., lógica, esquema de dados e interface HTML. A linguagem suporta desenvolvimento incremental, onde cada incremento é verificado estaticamente pelo sistema de tipos e introduzido na aplicação sem qualquer disrupção, garantindo a evolução segura e contínua da aplicação. Recorrendo a um grafo de dependências, a linguagem também possui propriedades reativas, mantendo o código e dados de uma aplicação permanentemente sincronizados. Esta reatividade permite manter as interfaces sempre atualizadas, i.e., sempre que os dados ou o código mudam, as interfaces são refrescadas para refletirem o novo estado da aplicação. Deste modo, o programador recebe *feedback* imediato do sistema, sabendo sempre qual o impacto do seu código na aplicação em execução.

A base da linguagem consiste em três operações básicas: `var`, `def` e `do`. A operação `var` é utilizada para declarar nomes que armazenem o estado da aplicação. A operação `def` associa uma transformação pura de dados a um nome. Por último, a operação `do` permite executar ações (definidas por intermédio da operação `action`), que por sua vez alteram o estado da aplicação.

De seguida, é exemplificada a criação de uma pequena aplicação *web* que demonstra algumas das capacidades da linguagem. O objetivo principal desta aplicação é a criação de um contador (lógica), que pode ser incrementado várias vezes. Além disso, pretende-se criar também uma página HTML (interface) com o intuito de apresentar o valor atual do contador, juntamente com um botão que permite incrementar o valor do contador. Por fim, serão apresentadas algumas

```
var counter = 0
var inc = action { counter := counter + 1 }
var plus5 = counter + 5
def times2 = counter * 2
```

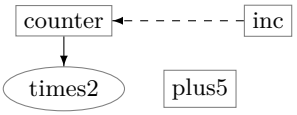


Figura 1. Exemplo de um contador.

reconfigurações à aplicação para acrescentar ou modificar funcionalidades, como mostrar outros valores ou definir manualmente o valor do contador.

A Figura 1 ilustra a criação do contador, recorrendo a um número inteiro. Na primeira linha do exemplo é definida uma variável de estado (`var`) `counter`, com o propósito de armazenar o valor (número inteiro) do contador. O nome `inc` denota uma ação que incrementa o contador (`counter`) numa unidade sempre que esta é executada. Note que, ao definir o nome `inc`, a ação fica definida mas a sua execução fica suspensa, podendo mais tarde ser executada explicitamente através da operação `do`. Com estes dois nomes fica definido o núcleo da aplicação. Considere-se, como exemplo, os nomes `times2` e `plus5` definidos na Figura 1, que guardam, respetivamente, o resultado da multiplicação da variável de estado `counter` por 2 e o resultado da soma entre o valor do nome `counter` e 5. Ao definir o nome `times2` como uma transformação pura de dados (`def`), este fica dependente da variável de estado `counter` e é recalculado (atualizado) sempre que o valor de `counter` muda. Este efeito é controlado através de um grafo de dependências entre os vários nomes. Através deste grafo é possível controlar quais os nomes (transformações puras de dados – `def`) que necessitam de ser atualizados, sempre que uma das suas dependências é modificada. Desta forma, sempre que uma das dependências de um nome muda, este é recalculado, de forma a refletir o estado mais atual do sistema.

Na Figura 1 ilustra-se o grafo das dependências correspondente ao contador, presente na mesma figura. Os nós retangulares da figura (variáveis de estado – `var`), que representam o estado persistente da aplicação, não são afetados pela propagação das alterações. Já os nós elípticos (transformações puras de dados – `def`), que representam a lógica da aplicação, são recalculados sempre que, pelo menos, uma das suas dependências muda. As setas contínuas representam uma dependência entre dois nomes, apontando no sentido da propagação. As setas a tracejado representam os nomes afetados pela execução de uma ação (através da operação `do`), não sendo consideradas na propagação de alterações.

Recorrendo à operação `do`, é possível executar a ação associada ao nome `inc` (`do inc`), modificando o valor associado ao nome `counter` de 0 para 1. Como o nome `times2` representa uma transformação pura de dados (`def`) que depende do nome `counter`, o seu valor é recalculado, passando a armazenar o valor 2. Contudo, `plus5` mantém-se inalterado, porque, sendo uma variável de estado, não é afetado pela propagação das alterações no grafo de dependências, continuando a armazenar o mesmo valor (5). Se a ação `inc` fosse novamente executada, o mesmo processo seria repetido, sendo que `counter` ficaria com o valor 2, `times2` ficaria com 4 e `plus5` manter-se-ia mais uma vez inalterado (5).

```
def counterPage =
  <div>
    <h1>"Counter"</h1>
    <p>counter</p>
    <button doaction=inc>
      "Increment"
    </button>
  </div>
```

Figura 2. Página HTML do contador com um botão.

Figura 3. HTML gerado pelo código da Figura 2.

```
def counterPage step =
    <div>
        <h1>"Counter"</h1>
        <p>counter</p>
        <button doaction=(action { counter := counter + step })>
            ("Add " ++ str step)
        </button>
    </div>
```

Figura 4. Página HTML do contador com um parâmetro.

Utilizando apenas as operações básicas apresentadas (`def`, `var` e `do`), é possível definir todos os aspetos de uma aplicação *web*, incluindo as páginas HTML. A Figura 2 ilustra um exemplo de como criar uma página HTML, denominada `counterPage`, que apresenta o valor atual do `counter`. O sistema trata os nomes que devolvem documentos HTML como *routes*. Assim, é possível aceder a esta página através do URL `/counterPage`.

Para permitir a interação do utilizador com o sistema, é possível associar ações a botões. Para tal, utiliza-se o atributo `doaction` do elemento `button`. A página `counterPage`, na Figura 2, contém um botão associado à ação `inc`. Sempre que o utilizador pressiona o botão, a respetiva ação `inc` é executada, alterando o `counter`. A página HTML gerada a partir do `counterPage` está representada na Figura 3.

As páginas HTML também seguem o modelo de propagação de alterações. Desta forma, sempre que uma página é redefinida ou uma das suas dependências é alterada, as alterações são propagadas para todos os utilizadores, garantindo que a página utilizada reflete os valores atuais do sistema. Por exemplo, caso o valor do contador seja alterado, o valor exibido na página (Figura 3) é atualizado sem a necessidade de refrescar toda a página.

A linguagem suporta ainda a criação de páginas que recebam parâmetros, tal como apresentado na Figura 4. Para visualizar à nova página `counterPage` basta aceder ao URL `/counterPage/<arg>`, onde `<arg>` será o número que ficará associado ao parâmetro `step`.

Comparativamente ao `counterPage` apresentado na Figura 2, o `counterPage` da Figura 4 é muito semelhante. Contudo há duas diferenças fundamentais. A

```
def counterPage =
    <div>
        <h1>"Counter"</h1>
        <p>"Counter: " counter</p>
        <button doaction=(inc)>
            "Increment"
        </button>
        <br/>
        <br/>
        "New value: " <input type="number" id="newCounter" value=counter/>
        <button doaction=(action { counter := #newCounter })>
            "Set counter"
        </button>
    </div>
```

Figura 5. Página HTML do contador com um *input*.

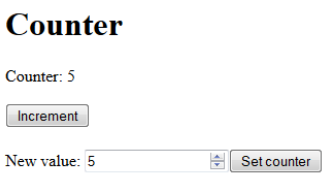


Figura 6. HTML gerado pelo código da Figura 5.

primeira está presente na primeira linha: o argumento `step`. Isto significa que `counterPage` não é apenas um valor HTML, mas sim uma função que que recebe um argumento (inteiro) e devolve uma página HTML. A outra diferença está na ação associada ao botão, que utiliza o valor passado como argumento para incrementar o `counter`. Tal como o `counterPage` definido anteriormente, o novo `counterPage` também é mantido sempre atualizado de forma a refletir os valores mais recentes das suas dependências. Por exemplo, se o utilizador aceder ao URL `/counterPage/10`, a página irá conter um botão onde se lê `"Add 10"`. Sempre que o utilizador premir o botão, o `counter` será incrementado em 10 unidades.

O mesmo exemplo poderia ser recriado recorrendo a um *input* fornecido pelo utilizador. A linguagem suporta a utilização de *inputs* HTML dentro de ações, como exemplificado na Figura 5, cuja interface HTML está representada na Figura 6. Como se pode observar, é utilizado um elemento *input* com o identificador `newCounter`. A ligação entre *inputs* na página e nomes na aplicação é feita pelo prefixo `#` (e.g. `#newCounter`). A utilização destes identificadores está restringida ao corpo das ações, pois só estas podem ter a sua execução adiada. É de realçar que, tal como o resto da linguagem, os *inputs* também são tipificados. No exemplo, o *input* `#newCounter` tem de ser do mesmo tipo do `counter`, sendo necessário adicionar o atributo `type` com a *string* `"number"`.

Na Figura 7 está representada uma nova reconfiguração do contador. A reconfiguração tem por objetivos apresentar no HTML os resultados de `plus5` e `times2`, e aumentar o incremento do contador para 2. Para tal, o nome `counterPage` é

```
var inc = action { counter := counter + 2 }

def plus5 = counter + 5

def counterPage =
    <div>
        <h1>"Counter"</h1>
        <p>"counter = " counter</p>
        <p>"counter + 5 = " plus5</p>
        <p>"counter * 2 = " times2</p>
        <button doaction=inc>
            "Increment"
        </button>
    </div>
```

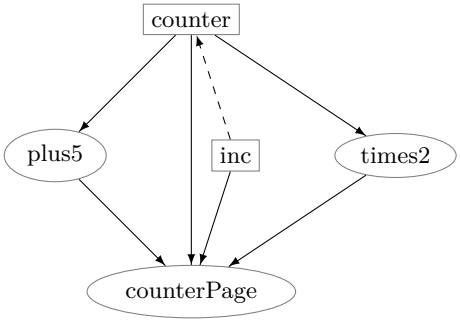


Figura 7. Reconfiguração do exemplo do contador.

```
def x = counter + 1
def y = x + 1
def x = y + 1
```

Figura 8. Dependência cíclica.

reconfigurado para incluir os valores de `plus5` e `times2`. A ação `inc` também é reconfigurada para incrementar o contador em 2 valores, ao invés de apenas 1. Por fim, o nome `plus5` é reconfigurado de **`var`** para **`def`**, de modo a ser recalculado sempre que o contador é incrementado.

O sistema de tipos garante, de forma estática, que a evolução das aplicações é feita de forma segura, sem causar disrupções ao sistema, ou seja, a introdução de incompatibilidade de tipos ou a inserção de ciclos no grafo de dependências (representado na Figura 7). Na Figura 8 está representado um exemplo em que a última instrução introduz um ciclo: os nomes `x` e `y` ficam a depender mutuamente um do outro. Isto faria com que a propagação de alterações entrasse num ciclo infinito, logo a instrução é rejeitada pelo sistema de tipos, não sendo introduzida no sistema. Um exemplo de uma mudança de tipo insegura seria a execução do seguinte comando: `var counter = "foobar"`. Uma vez que existem nomes que dependem de `counter` como número (inteiro), este não pode mudar de tipo para *string*. Portanto, neste caso, o sistema de tipos também rejeita este tipo de reconfigurações à aplicação.

3 Sistema de Suporte à Execução

O sistema de suporte à execução é constituído por 3 componentes principais:

- **Interpretador** – Interpreta, verifica estaticamente e executa o código das aplicações.
- **Base de dados** – Armazena persistentemente os dados e código das aplicações.

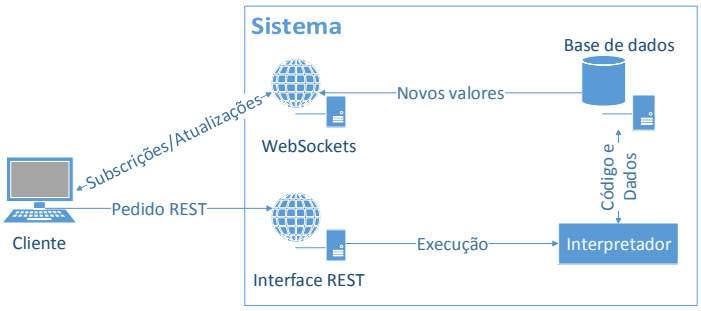


Figura 9. Arquitetura do sistema.

– **Servidor *web*** – Recebe pedidos dos clientes e envia atualizações via WebSockets.

O modo como estes componentes interagem entre si está representado na Figura 9. De seguida, descreve-se cada um dos componentes da arquitetura.

O interpretador é o componente central de todo o sistema, sendo responsável por implementar a linguagem apresentada na Secção 2. O interpretador executa o código de cada aplicação, verificando-o estaticamente através do sistema de tipos, garantido que cada operação permite evoluir a aplicação sem qualquer disrupção. Sempre que o interpretador altera o valor ou a expressão associada a um nome, o sistema utiliza o grafo de dependências para adicionar a uma fila todos os nomes que dependem do nome modificado. O sistema percorre a fila e recalcula os novos valores de cada um dos nomes. Quando a fila fica vazia, significa que as alterações foram propagadas a todo o sistema. Estas recomputações são processadas sequencialmente. Um melhoramento futuro passa por paralelizar as recomputações dos nomes que não têm dependentes em comum.

A base de dados (MongoDB[1]) é utilizada para armazenar, persistentemente, o código e dados das aplicações. Várias informações relativas a todos os nomes declarados na aplicação são armazenadas na base de dados, como a AST, o último valor calculado (evitando recomputações desnecessárias), o tipo da expressão (utilizada pelo sistema de tipos na verificação estática dos incrementos de código) e as dependências associadas a esse nome (de modo a propagar alterações pelo grafo de dependências). Tanto o código como os dados são guardados na mesma base de dados de igual forma, recorrendo à serialização das AST e dos resultados, de modo a serem armazenados como *strings*. Sempre que se pretende obter código ou dados da base de dados, é feita a desserialização dessas *strings*.

O servidor *web* é o responsável por receber os pedidos dos clientes e gerar páginas HTML. Este servidor está dividido em duas interfaces: REST e WebSockets. A interface REST é utilizada sempre que a comunicação é iniciada pelo cliente. O método `GET` é usado para obter o valor associado a um nome. Caso esse nome seja uma função, os argumentos podem ser passados no URL. Por exemplo, caso se pretenda chamar uma função, de nome `page`, que recebe dois

[1] `https://www.mongodb.org`

```
def counterPage =
    let step = 2 in
        <div>
            <h1>"Counter"</h1>
            <p>"Counter: " counter</p>
            "New value: " <input type="number" id="newCounter" value=counter/>
            <button doaction=(action { counter := #newCounter })>
                "Set counter"
            </button>
            <br/>
            <button doaction=(action { counter := counter + step })>
                "Increment"
            </button>
        </div>
```

Figura 10. Página HTML do contador com duas ações.

números, o endereço seria `page/1/2`, onde 1 é o primeiro argumento da função e 2 o segundo. O método `POST` é utilizado pelo programador para adicionar novo código à aplicação, sendo esse código passado no corpo do pedido. Por fim, o método `PUT` é utilizado para executar ações que, tipicamente, estão associadas a botões ou outros eventos. Para tal, é necessário incluir no corpo do pedido o identificador da ação, o ambiente em que deve ser executada e os *inputs* preenchidos pelo cliente que sejam usados pela ação. A interface WebSockets é usada para enviar ao cliente os novos valores associados a um nome. Para tal, utiliza-se o padrão do observador, onde o cliente inicialmente subscreve a um conjunto de nomes, dos quais pretende receber sempre os valores mais atuais. Sempre que um desses nomes muda, o servidor envia ao cliente uma notificação com o novo valor, atribuindo propriedades reativas às aplicações. Por exemplo, quando o utilizador carrega a página representada nas Figuras 2 e 3, o *browser* envia, por WebSockets, um pedido a subscrever ao nome `counterPage`. Quando o contador é incrementado, a página é recalculada para exibir o novo valor. Consequentemente, o servidor envia a todos os subscritores da página `counterPage` uma mensagem a indicar qual o novo valor do elemento HTML que contém o contador.

De seguida, na Secção 3.1 é explicado em detalhe como é efetuado o processo de *hoisting* de nomes, que permite que funções parcialmente avaliadas sejam utilizadas nas páginas HTML. Na Secção 3.2 aborda-se como as páginas HTML são mantidas em consonância com os dados presentes no servidor. Por último, na secção 3.3, é apresentado o IDE usado para o desenvolvimento de aplicações.

3.1 *Hoisting*

A linguagem base suporta funções de primeira ordem com avaliação parcial. Também as ações contêm *closures* que podem ter a sua execução adiada e que estão, tipicamente, associadas a botões e outros eventos do cliente. Quando o utilizador pretende executar uma ação, envia um pedido ao servidor indicando qual o nome dessa ação, logo cada ação deve estar associada a um nome único. Contudo, esta ação pode estar definida *inline* (ver Figura 10), não estando

```
var counterPage$1 = action { counter := #newCounter }

var counterPage$2 = action { counter := counter + step }

def counterPage =
    let step = 2 in
        <div>
            <h1>"Counter"</h1>
            <p>"Counter: " counter</p>
            "New value: " <input type="number" id="newCounter" value=counter/>
            <button doaction=(counterPage$1)>
                "Set counter"
            </button>
            <br/>
            <button doaction=(counterPage$2)>
                "Increment"
            </button>
        </div>
```

Figura 11. Programa da Figura 10 após o processo de *hoisting*.

associada a nenhum nome único. Para resolver este problema, é preciso garantir que todas as ações têm um nome único associado. Isto pode ser alcançado recorrendo à técnica padrão de *hoisting*, i.e., colocar ações definidas *inline* num nome *top-level* e substituir a sua utilização por um apontador. Esta técnica é, normalmente, utilizada pelos compiladores de linguagens funcionais.

De seguida, apresenta-se um exemplo de como a linguagem lida com o *hoisting* de ações. Na Figura 10 está uma nova reconfiguração do `counterPage`. Esta reconfiguração tem por objetivo apresentar dois botões: um para definir o valor do contador e outro para incrementá-lo. Salienta-se o facto de ambos os botões recorrerem a ações definidas *inline*, sendo que o botão de incremento também utiliza valores de escopo. Quando o processo de *hoisting* é aplicado, este programa é transformado para o que está representado na Figura 11. O código HTML correspondente tem o aspeto ilustrado na Figura 12. Quando o utilizador prime um dos botões, o *browser* envia um pedido ao servidor indicando qual das ações pretende executar, bastando para isso indicar o identificador da ação.

No caso da página da Figura 12, essa ação pode ser `counterPage$1` ou `counterPage$2`. Caso a ação necessite de valores de ambiente para ser executada, como é o caso da ação `counterPage$2`, esses serão incluídos no objeto `env`, como representado na Figura 12. Assim, quando o utilizador indica ao servidor a intenção de executar uma ação que depende do ambiente, os valores necessários para a sua execução são enviados no corpo do pedido. Caso a ação utilize valores de *input*, tal como a ação `counterPage$1`, os nomes que são necessários enviar são indicados no vetor `args`. Desta forma, o *browser* sabe quais são os *inputs* que é necessário enviar ao servidor sempre que pretende executar uma determinada ação. Quando o botão associado à ação `counterPage$1` é pressionado, o cliente envia

```
<div>
    <h1>Counter</h1>
    <p>Counter: 0</p>
    New value: <input type="number" id="newCounter" value="0"/>
    <button doaction={
        "action": "counterPage$1",
        "env": {},
        "args": ["newCounter"]
    }>
        Set counter
    </button>
    <br/>
    <button doaction={
        "action": "counterPage$2",
        "env": {"step": 2},
        "args": []
    }>
        Increment
    </button>
</div>
```

Figura 12. HTML gerado pelo programa 11.

```
def f x =
    let a = 1 in
        y => x + y + a
```

Figura 13. Função que recebe dois parâmetros.

```
var f$1 = x + y + a

var f$2 =
    let a = 1 in
        <y, f$1, [a = 1]>

def f = <x, f$2, []>
```

Figura 14. Função da Figura 13 após o processo de *hoisting*.

ao servidor um pedido contendo o identificador da ação e os *inputs* necessários (o valor introduzido pelo utilizador no *input* com id `newCounter`).

***Hoisting* de funções.** O processo de *hoisting* também é aplicado ao corpo das funções. Na linguagem, as funções são de primeira ordem, podendo ser retornadas como valores e executadas mais tarde. Para tal, é preciso aplicar o processo de *hoisting* no corpo das funções, de modo a que os resultados não contenham código. Um exemplo de *hoisting* de funções está representado nas Figuras 13 e 14, onde se declara uma função `f` com o parâmetro `x` que devolve uma função anónima com o parâmetro `y`. Realça-se o facto de a função anónima de parâmetro `y` utilizar um nome de escopo definido fora do seu corpo. As funções são convertidas para tuplos com três elementos: o nome do argumento, o nome que contém o corpo da função e o ambiente da função. O nome `f` passa a conter um tuplo que indica qual o nome do argumento (neste caso, `x`) , qual o nome que contém o corpo da função

```
var empty = true

def page =
    if empty then
        <strong>"empty"</strong>
    else
        <em>"full"</em>
```

Figura 15. Página HTML com dois *templates* possíveis.

(neste caso, `f$2`) e qual o ambiente em que foi declarado (neste caso, o ambiente é vazio) O nome `f$2` contém o corpo da função de parâmetro `x`, devolvendo um tuplo correspondente à segunda função. A principal diferença deste tuplo e do anterior é a presença do ambiente, contendo o valor associado ao nome `a`. O corpo da segunda função está associado ao nome `f$1`. Os tuplos são tratados como apontadores para código. Sempre que se depara com um destes tuplos, o interpretador adiciona uma nova entrada ao ambiente da função (o terceiro elemento do tuplo), associando o nome do parâmetro (o primeiro elemento do tuplo) ao valor passado na aplicação da função. De seguida, utiliza este ambiente na execução do código associado ao nome que contém o corpo da função (o segundo elemento do tuplo).

3.2 Refrescamento de páginas

Todas as páginas mantêm uma ligação por WebSockets com o servidor, de modo a receberem atualizações caso algum valor presente na página seja atualizado. O servidor mantém uma lista de subscrições, contendo informações sobre os utilizadores e em que páginas se encontram. A subscrição de uma página guarda as dependências associados ao seu cálculo. Sempre que uma dessas dependências é alterada, a página é recalculada. Estas dependências são calculadas com base no sistema de tipos da linguagem, com a única diferença de serem armazenadas na subscrição da página e não na base de dados utilizada pelo interpretador.

Para evitar que o servidor reenvie ao cliente toda a página sempre que esta é modificada, uma subscrição também guarda o último valor calculado. Assim, quando a página sofre uma modificação, basta comparar o novo valor com o antigo e enviar apenas as alterações. Na versão atual do protótipo, esta comparação é feita apenas nos elementos HTML de mais alto nível. Caso a página utilize parâmetros passados no URL, estes também são guardados na subscrição.

Por fim, a subscrição guarda um identificador único associado ao elemento base da página HTML. Este identificador serve apenas como um auxiliar para se saber quando é desnecessário utilizar o algoritmo de comparação, sendo preferível reenviar toda a página. A Figura 15 ilustra esta situação, onde página `page` pode tomar dois *templates* distintos. Cada um destes *templates* HTML está associado a um identificador único. Quando o valor do nome `empty` é `true`, o nome `page` contém a *string* `empty` a negrito. Quando o nome `empty` muda para `false`, o nome `page` fica associado a um *template* diferente, com a *string* `full` a itálico.

Figura 16. IDE disponível através do *browser*.

3.3 Ambiente de Desenvolvimento

O nosso servidor [9] fornece um ambiente de desenvolvimento (IDE) que permite ao programador desenvolver aplicações *web* diretamente no *browser*. O IDE, apresentado na Figura 16, divide-se em três painéis:

1. **Editor** – É neste painel onde o programador pode, efetivamente, desenvolver as aplicações *web*, escrevendo código que será interpretado pelo servidor.
2. **Painel de resultados** – Este painel exibe os resultados retornados pelas expressões submetidas ao servidor. Este painel e o editor atuam como um *read–eval–print loop* (REPL), permitindo ao programador rapidamente experimentar código.
3. **Painel de nomes** – O painel de nomes tem o propósito de mostrar ao programador o estado atual da aplicação. Para tal, contém a lista de nomes definidos e o respetivo código e/ou resultados (as variáveis de estado – `var` – apenas mostram o valor corrente). Cada nome presente no painel dispõe de um conjunto de botões que executam diferentes operações. Por exemplo, é possível executar uma ação ou abrir uma página HTML diretamente no editor (como demonstrado na Figura 16 para o elemento `page`).

No seu estado atual, o IDE permite alcançar o objetivo de aproximar o programador da aplicação final, no caso de aplicações de pequena dimensão. Contudo, para facilitar o desenvolvimento de aplicações de maior dimensão, várias modificações teriam de ser feitas. Algumas dessas modificações são, por exemplo, o suporte a um editor de texto mais sofisticado, permitir separar os nomes por diversos módulos, editor de CSS e JavaScript, etc. Estes melhoramentos fazem parte do trabalho futuro do projeto.

4 Trabalho Relacionado

Há várias ferramentas e linguagens que implementam as funcionalidades presentes descritas neste artigo, como a reconfiguração dinâmica, a propagação de alterações e a reatividade. Nesta secção apresentamos dois modelos utilizados para implementar tais funcionalidades: *dynamic software updates* e programação incremental de *data-flow*.

Dynamic Software Updates. Uma forma de efetuar a reconfiguração dinâmica de sistemas é através do uso de *dynamic software updates* (DSU) [7]. Os DSU permitem modificar código em execução através da aplicação de *patches*, sem a necessidade de interromper o sistema. Kitsune [5] e Ekiden [6] são dois exemplos de ferramentas que seguem o modelo de DSU. Estas ferramentas não eliminam as fases de compilação e de *deployment*. O *patch* a ser aplicado precisa de ser compilado utilizando um compilador modificado para o efeito. Também a aplicação do *patch* precisa de ser controlada pelo programador, ao adicionar à aplicação código de suporte a reconfigurações dinâmicas durante a sua execução. Por vezes, também é necessário código para lidar com o processo de evolução da aplicação, onde se define como os dados devem ser traduzidas de uma versão do código para a seguinte. Esta evolução nem sempre é verificada estaticamente pelo compilador, o que pode resultar na introdução de novos *bugs* durante o processo de reconfiguração. Todos estes aspetos são tratados implicitamente pelo nosso sistema, que assegura a evolução segura do código e dados.

***Programação Incremental de* Data-Flow.** A nossa linguagem segue o modelo de programação incremental de *data-flow*, permitindo construir uma aplicação em pequenos incrementos que são introduzidos no sistema em execução sem interrupções. As linguagens que se enquadram nesta categoria mantém um grafo de dependências entre os nomes definidos no programa, de modo a propagar as alterações para todos os nomes. O grafo pode ser utilizado para conceder propriedades reativas às aplicações. Esta técnica é utilizada por linguagens como o SuperGlue [10] ou o Circa [3] para implementar ambientes de programação *live*, onde a aplicação em execução está sempre sincronizada com o código presente no editor, fornecendo ao programador *feedback* imediato sobre o seu trabalho, tal como no nosso projeto. Uma particularidade do Circa é o suporte a dois modos diferentes: textual e visual. Estes modos estão constantemente sincronizados, sendo possível fazer modificações ao programa através do modo visual (utilizando técnicas *drag and drop*) que são refletidas no código. A principal diferença entre estes projeto e o nosso é o facto de não se focarem em aplicações *web*, mas sim programas com o intuito de correrem localmente.

O LightTable [4] é um IDE que também aplica conceitos de programação incremental e de *data-flow* a linguagens mais tradicionais, como JavaScript, Python e Clojure. O IDE mantém a aplicação a correr em sintonia com o código presente no editor, de modo similar ao SuperGlue, Circa ou ao nosso projeto. Uma diferença do LightTable é também apresentar o valor corrente de uma variável diretamente no editor de texto. Tal como no caso do Circa e do SuperGlue, o

LightTable não está idealizado para o desenvolvimento de aplicações *web*, o que não permite evoluir uma aplicação em produção sem quebras de serviço.

5 Trabalho Futuro

O projeto apresentado neste artigo ainda se encontra em desenvolvimento, havendo funcionalidades importantes ainda por implementar, que serão apresentadas e discutidas nesta secção.

Múltiplas Aplicações. Uma das funcionalidades por implementar é o suporte para várias aplicações. Atualmente, o sistema suporta apenas uma aplicação. Com esta funcionalidade, seria possível correr diferentes aplicações, de forma isolada, no mesmo sistema. Estas aplicações poderão ser criadas como clones de outras. Isto permite ao programador lidar com aplicações de forma similar aos *branches* existentes no sistema de controlo de versões Git. Por exemplo, o programador pode fazer clone de uma aplicação, testar modificações no clone e, posteriormente, fazer *merge* do clone com a aplicação original. Esta funcionalidade pode ser útil quando o programador pretende simular a existência de um servidor de desenvolvimento, de modo a testar alterações sem as tornar visíveis a todos os utilizadores.

Valores de Sessão. Na versão atual do sistema, o estado é partilhado entre todos os utilizadores da aplicação. Isto previne a aplicação de guardar dados distintos para cada cliente (por exemplo, o nome de utilizador). Com valores de sessão, seria possível ter várias versões de um mesmo nome, i.e., o nome toma um valor diferente para cada cliente, o que aumentaria o leque de possibilidades do sistema. Por exemplo, é possível criar aplicações que possuam sistemas de *login*, mostrando informações diferentes a utilizadores distintos.

Persistência. Outra funcionalidade que pretendemos oferecer é a possibilidade de mapear dados diretamente na base de dados. No sistema atual, tanto dados como código são guardados na mesma base de dados, através da serialização das AST e dos resultados. Com esta funcionalidade, seria possível interagir diretamente com a base de dados, sem recorrer à serialização de objetos. Isto permitiria que o programador criasse tabelas de mais alto nível e efetuasse pesquisas sobre os dados de um modo muito mais eficiente.

JavaScript e CSS. De momento, o sistema apenas permite incluir código JavaScript e estilos CSS nas aplicação por intermédio dos atributos dos elementos HTML. Para permitir o desenvolvimento de aplicações mais complexas, a inclusão JavaScript e CSS é uma necessidade. A solução preferencial seria estender a linguagem atual com operações que permitam definir o estilo e o comportamento do lado do cliente da aplicação.

6 Conclusão

Neste artigo apresentamos um sistema de desenvolvimento de aplicações *web* com base numa linguagem incremental e reativa [2]. Como exemplificado nas Secções 2 e 3, a linguagem tem suporte para a definição de toda uma aplicação *web*: lógica, esquema de dados e interface. Combinando elementos de linguagens de *back-end* e de *front-end*, o sistema permite criar, em poucas linhas, aplicações *web* completas (desde a lógica do servidor à do cliente) com propriedades reativas.

O ambiente de desenvolvimento permite ao programador estar, permanentemente, consciente de qual o comportamento da aplicação *web* num ambiente de produção, ao combinar programação incremental com reatividade. Com o *deployment* implícito da aplicação e a possibilidade de a experimentar à medida que é implementada, o ciclo de evolução é encurtado e o programador consegue receber informação importante sobre o comportamento da aplicação mais cedo no ciclo. Acreditamos que ao fornecer *feedback* constante e imediato ao programador sobre o seu trabalho conduz a uma maior produtividade. Ao executar as reconfigurações sem a necessidade de reiniciar ou interromper o sistema, evita-se quebras de serviço durante o processo de evolução da aplicação.

Ao implementar as tarefas descritas na secção 5, as potencialidades oferecidas pelo sistema crescem, ao permitir utilizar funcionalidades habitualmente presentes em *frameworks web*, como sessões e armazenamento em base de dados.

Referências

1. Chen, Y., Dunfield, J., Acar, U.A.: Type-directed automatic incrementalization. In: ACM SIGPLAN Notices. vol. 47 (2012)
2. Domingues, M., Seco, J.C.: Typeful updates on reactive live web programming. Tech. rep. (2013)
3. Fischer, A.: Introducing circa: A dataflow-based language for live coding. In: Live Programming (LIVE), 2013 1st International Workshop on. IEEE (2013)
4. Granger, C., Attorri, R.: Light table, `http://lighttable.com/`
5. Hayden, C.M., Smith, E.K., Denchev, M., Hicks, M., Foster, J.S.: Kitsune: Efficient, general-purpose dynamic software updating for c. In: ACM SIGPLAN Notices. vol. 47 (2012)
6. Hayden, C.M., Smith, E.K., Hicks, M., Foster, J.S.: State transfer for clear and efficient runtime updates. In: 2011 IEEE 27th ICDEW (2011)
7. Hicks, M., Moore, J.T., Nettles, S.: Dynamic software updating. In: Proceedings of the ACM SIGPLAN 2001 Conference on PLDI (2001)
8. Lin, D.Y., Neamtiu, I.: Collateral evolution of applications and databases. In: Proceedings of the IWPSE-EVOL. ACM (2009)
9. Mateus, J., Domingues, M., Seco, J.C., Lopes, T., Martins, N.: Live programming prototype, `https://live-programming.herokuapp.com/`
10. McDirmid, S.: Living it up with a live programming language. In: ACM SIGPLAN Notices. vol. 42 (2007)

MODUS: uma metodologia de prototipagem de interfaces baseada em modelos

Marina Machado, Rui Couto, and José Creissac Campos

HASLab/INESC TEC & Departamento de Informática/Universidade do Minho
Campus de Gualtar, 4710-057 Portugal
pg25336@alunos.uminho.pt, jose.campos@di.uminho.pt, ruicouto@di.uminho.pt

Resumo A interface de utilizador é essencial para o sucesso de uma aplicação, ora a sua implementação requer tempo e esforço. Metodologias baseadas em modelos suportadas por uma geração automática têm sido propostas como solução para reduzir os custos de desenvolvimento. No entanto, um elevado nível de automação nestas abordagens requer o uso de modelos detalhados da interface, originando complicações. Por um lado, esta perspetiva é contrária a uma conceção baseada no progressivo refinamento de *mockups*, típica do desenvolvimento de interfaces. Por outro lado, obriga uma distinção entre os modelos utilizados na lógica de negócio e na interface. Este artigo propõe uma abordagem para gerar a interface de utilizador baseada diretamente em modelos estruturais da lógica de negócio. A identificação do domínio de aplicação é um factor chave da metodologia, permitindo automatizar o processo de geração. Por sua vez a separação entre conteúdo e forma favorece o refinamento iterativo das interfaces geradas.

Palavras Chave: Geração Automática de Interfaces de Utilizador, *Model-driven engineering*, *Model-Based User Interface Development*

1 Introdução

Na sociedade moderna um vasto leque de pessoas tem acesso à tecnologia, estando acostumado a usá-la no seu dia a dia. Sendo a interface de um sistema o único meio de comunicação com o utilizador, esta torna-se num fator decisivo para o sucesso das aplicações [1].

Estudos realizados na década de 1980 e 1990 mostraram que quase 50% do esforço de desenvolvimento de uma aplicação é dedicado à *Interface de Utilizador* (IU). A crescente complexidade das interfaces requer soluções de desenvolvimento inovadoras, dificultando o seu processo de criação. Tudo leva a crer [2] que o tempo gasto na sua implementação continua significativo, sendo a sua redução uma forma de otimizar o desenvolvimento das aplicações [3,4,5].

Diversos tipos de ferramentas têm sido criados para facilitar e acelerar a geração de interfaces. Na maioria dos casos necessitam da enunciação explícita de cada elemento gráfico, como por exemplo as ferramentas de especificação gráfica e as baseadas em linguagens. Um processo de criação dependente exclusivamente

destas perspetivas permanece lento e penoso [1,6,7,8]. Todavia, as ferramentas baseadas em modelos destacam-se por fornecer um certo grau de automação ao processo de criação, reutilizando esforço prévio de desenvolvimento. A utilização destas ferramentas tem sido explorada como potencial solução do problema, dando origem ao *Model-Based User Interface Development* (MBUID).

Questiona-se então o porquê das ferramentas baseadas em modelos não se terem tornado a norma no desenvolvimento automatizado de IU. Nesta categoria o nível de automação tende a ser reduzido. Um aumento no grau de automação é geralmente traduzido pela inclusão de informação adicional por parte do utilizador da ferramenta, complexificando a elaboração dos modelos [2]. As ferramentas que focam a automação falham em diversos aspetos na produção das interfaces. Em particular, tendem a ser criticadas pela fraca integração no processo de desenvolvimento de software. Por um lado, uma geração baseada em modelos tende a ter problemas na integração com o processo criativo dos *designers*, tendo um impacto negativo na qualidade da IU gerada [9]. Por outro, os modelos utilizados tendem a divergir dos modelos da camada de negócio usados em abordagens *Model-driven engineering* (MDE), levantando problemas de coordenação e produtividade [2,10].

Este artigo apresenta uma nova abordagem, suportada pela ferramenta *MODUS* (*MOdel-based Developed User Systems*), propondo a seguinte lista de contribuições:

- Tira partido da separação entre a definição do conteúdo e aspeto gráfico da interface (neste caso, para aplicações Web) de forma a gerar "esqueletos", os quais poderão ser refinados pelos *designers*.
- Centra-se no uso de um modelo estrutural do sistema, um diagrama de classes em *Unified Modelling Language* (UML) [11], promovendo a integração entre as abordagens MDE e MBUID. O diagrama de classes por definição enuncia todas as classes e respetivas relações necessárias para o desenvolvimento do sistema, sendo fundamental num contexto de MDE. Na abordagem proposta, aparenta ser uma opção apropriada pois as entidades que o compõem são decisivas no conteúdo da aplicação, sendo específicos à mesma.
- Tira partido da definição prévia do domínio da aplicação para limitar a necessidades de modelos adicionais no processo de criação, possibilitando um processo de geração substancialmente mais automatizado. Com efeito, para o mesmo domínio de aplicação, constata-se que as interfaces de utilizador tendem a ser semelhantes em navegação, estrutura e componentes visuais.
- Nesta abordagem o conceito de *Evolutionary Prototyping* [12] é indispensável. O utilizador da ferramenta tem a capacidade de manipular o resultado em qualquer uma das etapas do processo de geração, o qual será refinado até a obtenção da interface final. Graças ao seu cariz iterativo a interface elaborada satisfaz um maior número de requisitos do utilizador.
- Integra a possibilidade de seleção de detalhes de aparência da interface, tal como *front end frameworks*, *responsive design* para diferentes dispositivos, *templates*, entre outros. Deste modo, o utilizador terá um maior controlo sobre o aspeto visual do resultado final da geração.

O artigo está organizado segundo a seguinte estrutura. A secção 2 introduz os conceitos de base e a secção seguinte 3 aborda trabalhos relacionados ao contexto da abordagem. Posteriormente é apresentada a metodologia proposta e a ferramenta que a suporta na secção 4 e 5, respetivamente. Por fim, a secção 6 apresenta conclusões e o trabalho futuro a ser realizado.

2 Contextualização

2.1 *Model-Driven Engineering*

Os modelos, na engenharia de software, permitem que os programadores analisem o sistema antes de iniciar a implementação, concretizando idealmente a especificação que levará à solução final. Ao longo do tempo esta tendência tornou-se para muitos a norma, dando origem ao paradigma de MDE [13,14].

Nesta metodologia, modelos de alto nível de abstração, que representam o problema que o sistema enfrenta, são convertidos sucessivamente em modelos de nível inferior, com o objetivo de os transformar em sistemas executáveis [13,14]. Esta conversão pode ser manual ou automatizada. Em ambas abordagens a utilização de modelos permite assegurar a coerência no desenvolvimento do software, garantindo uma melhor qualidade e correção do resultado final. Acima de tudo, ao recorrer a uma transformação automática dos modelos para o código executável, assegura-se uma redução do tempo de desenvolvimento [10,15].

MDE é tipicamente aplicada à criação das camadas da lógica de negócio e de dados [16,17]. Todavia os modelos elaborados neste contexto poderiam ser reutilizados para a geração da IU, sendo uma das premissas da abordagem proposta.

2.2 *Model-Based User Interface Development*

MBUID tende a ser visto como MDE de interfaces de utilizador. É uma metodologia que procura reduzir a quantidade de tempo e esforço despendida no desenvolvimento da IU, garantindo a qualidade da interface produzida [10,15,18].

A *Cameleon Reference Framework*, amplamente aceite como a arquitetura de referência para MBUID, especifica quatro níveis principais de modelação[19,20,21]:

1. **Modelo de Domínio e Tarefas** - descrição das tarefas do utilizador e dos conceitos de domínio relacionados com a sua realização.
2. **Abstract User Interface (AUI)** - descrição da interface em termos de *Abstract Interaction Units* ou *Abstract Interaction Objects* e respetivas relações. AUI é independente da tecnologia e modalidade de interação.
3. **Concrete User Interface (CUI)** - descrição da interface em termos de *Concrete Interaction Units* ou *Concrete Interaction Objects*, definindo o *layout* e a navegação. Sendo a CUI dependente da modalidade de interação, descreve o *Look & Feel* da aplicação.
4. **Final User Interface (FUI)** - descrição da interface em termos do código fonte, quer numa linguagem de programação ou de *mark-up*. A FUI pode ser interpretada ou executada, após compilação do código.

A geração em MBUID baseia-se em modelos declarativos de alto nível. No entanto é frequente usarem-se modelos detalhados sobre diferentes aspetos da interface. Este facto limita a aceitação da abordagem, pois existe um custo acrescido de modelação, quer em número de modelos, quer em esforço de elaboração [10,15,18]. Decidiu-se, assim, centrar a abordagem proposta num processo de geração baseado num modelo estrutural do sistema complementando com a especificação do domínio da aplicação. Este diagrama é a base da criação, parcial ou completa, da IU, podendo ser refinada com apoio de ferramentas adequadas.

3 Trabalho Relacionado

Diversas abordagens baseadas em modelos têm vindo a ser desenvolvidas, tentando responder aos desafios trazidos pela evolução das interfaces de utilizador. Podem ser identificadas quatro gerações de ferramentas MBUID.

As ferramentas da primeira geração baseavam-se num modelo declarativo universal, apoiando-se numa criação totalmente automatizada. O foco era a geração de aplicações *Desktop* baseadas em operações *Create, Read, Update and Delete* (CRUD), como as apresentadas em [22,23,24]. As interfaces tendiam a ser simples, seguindo sempre o mesmo modelo visual.

A segunda geração define o modelo da IU como uma composição de modelos declarativos. A criação da interface é um processo cumulativo, procurando melhorar a qualidade da IU à custa da especificação de modelos adicionais. Nestas ferramentas o nível de automação tende a ser baixo, sendo que um aumento no grau de automação implica a inclusão de informação auxiliar sobre a interface.

A terceira geração foca-se nos desafios trazidos pelo aparecimento de novos dispositivos e plataformas. *TERESA* [25] é um exemplo típico desta geração. Para solucionar os problemas a especificação é dependente do contexto. Para cada contexto suportado, é necessário definir novas versões dos modelos. Para IUs de aplicações web, o uso de tecnologias *front end* como *Javascript* e *CSS3* permitem aumentar significativamente a compatibilidade entre diferentes dispositivos e plataformas em tempo de execução [26]. A integração dessas tecnologias no processo de geração poderá ser uma alternativa à especificação do contexto.

A quarta geração centra-se na elaboração de interfaces sensíveis ao contexto, através do desenvolvimento *multi-path* [27] e na integração com *web services*. A proposta apresentada é em grande medida ortogonal a estas preocupações, focando-se, nesta fase, em fornecer uma solução eficiente para a criação de interfaces, tendo como ponto de partida modelos estruturais da camada de negócio.

Os desenvolvimentos verificados em MBUID têm vindo a ser aplicados na indústria de software. Ferramentas foram elaboradas adotando alguns conceitos mais estabilizados deste paradigma. Exemplos são *Outsystems* [28] e *Integranova* [29]. *Outsystems* pretende aumentar a produtividade dos programadores. Embora crie uma IU a partir de um modelo de base de dados, o nível de automação é relativamente baixo, sendo a interface construída num editor *What You See Is What You Get* (WYSIWYG). Por sua vez, *Integranova* visa a criação de todas

as camadas da aplicação. Gera uma IU baseada em CRUD com um alto nível de automação, sem possibilidade de melhoria durante o processo.

De um modo geral, constata-se que as ferramentas mais automatizadas necessitam da inclusão de informação adicional, gerando interfaces mais rígidas e menos apelativas. As ferramentas mais flexíveis requerem uma maior intervenção do utilizador, não resolvendo a questão dos custos de desenvolvimento. A abordagem proposta neste artigo tenciona conciliar ambos aspetos, procurando contribuir para a solução do problema.

4 Abordagem

A Figura 1 resume a abordagem apresentada neste artigo. Em termos gerais, o processo interpreta o *input* do utilizador para gerar a IU, sob a forma de uma interface *Web*. No entanto, a abordagem contém etapas intermédias. Na Figura 1 cada etapa está etiquetada numericamente, sendo o fluxo representado por setas. Os dados de entrada estão etiquetados alfabeticamente, em maiúsculas os fornecidos pelo utilizador (específicos de uma interface em concreto) e em minúscula os disponibilizados pela ferramenta (de uma aplicação genérica).

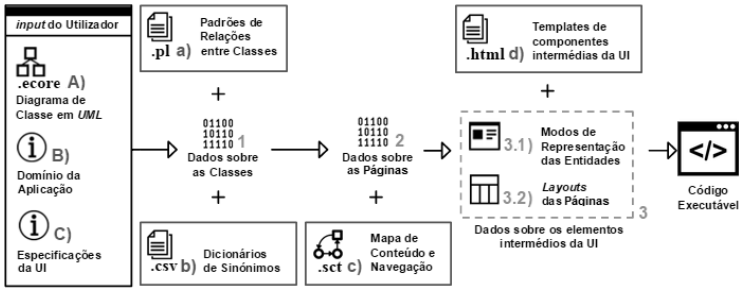


Figura 1: Visão global da abordagem

O processo é iniciado com (Figura 1, *input* do Utilizador): A) o diagrama de classes em UML no formato *ecore* [30] (ver exemplo na Figura 2); B) o domínio da aplicação, o qual é explicitado na ferramenta; C) informação adicional sobre a geração da IU, como por exemplo a *front end framework* a utilizar, a qual é definida na ferramenta. Estes dados são essenciais para todo o processo.

A geração da interface é composta por três etapas intermédias. Na primeira etapa interpretam-se os dados de entrada através de a) padrões arquitecturais e b) bases de conhecimento semântico, de forma a identificar entidades relevantes no modelo. Na segunda etapa definem-se as páginas da interface, a nível de conteúdo e navegação, a partir de c) um modelo abstrato da interface para o domínio inicialmente definido. Na terceira etapa procede-se à concretização das componentes intermédias da interface, nomeadamente modos de representação de entidades e *layouts* das páginas, através do uso de d) *templates*.

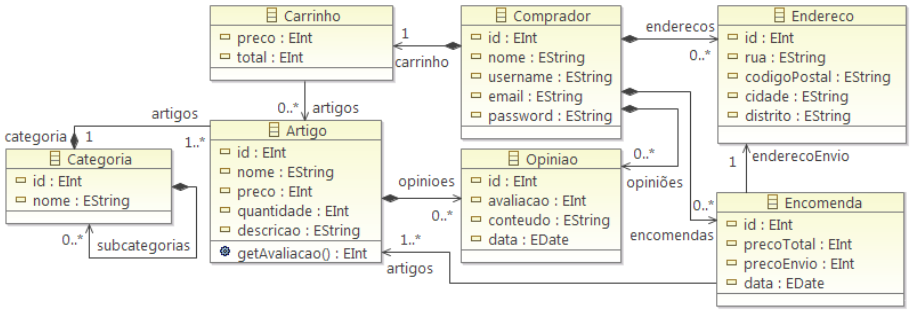


Figura 2: Diagrama de Classes em UML de uma aplicação web do tipo *eCommerce*

4.1 Reconhecimento das Classes *Standard*

Uma entidade, por outras palavras, uma classe do diagrama de classes, pode ser associada a uma classe *standard*. As classes *standard* correspondem a classes que usualmente existem na modelação de um dado domínio de aplicação. Um exemplo seria associar a entidade `Artigo` da Figura 2 à classe `Product` no domínio `eCommerce`. Esta análise permite estabelecer a base de muitas das decisões tomadas sobre o desenho da interface ao longo do processo de criação. O processo de associação é efetuado em duas etapas. A primeira etapa consiste na identificação de padrões arquiteturais em que as classes *standard* tipicamente estão envolvidas. A segunda etapa resume-se à estimativa do grau de compatibilidade entre o nome da entidade identificada e o da classe *standard*.

Na primeira etapa, transforma-se o conhecimento presente no diagrama de classes numa ontologia *Prolog*. Optou-se por usar uma linguagem de programação lógica de modo a inferir sobre a base de conhecimento da ontologia extrapolada. Desta forma, a informação das entidades é expressa em factos, pois são predicados que representam o diagrama. Por sua vez, cada padrão é descrito numa regra, a qual é consultada para obter a veracidade da correspondência com cada uma das entidades. Este processo baseia-se na abordagem apresentada em [32].

A base de conhecimento é composta por dois tipos de factos, representados na Listagem 1.1. O facto da linha 1 enuncia a existência de uma entidade com o nome `E` (`Entity`). O facto da linha 2 especifica a existência de uma relação entre entidades, sendo: `R` (`Relation`) o nome da relação; `F` (`From`) a entidade de partida; `T` (`To`) a entidade de chegada; `L` (`Lower Bound`) o limite inferior e, por fim, `U` (`Upper Bound`) o limite superior da cardinalidade de `T` na relação. Um exemplo é apresentado na Listagem 1.2, no qual existem duas entidades, Comprador e Endereço, em que um comprador pode ter múltiplos endereços.

```
1 entity(E).
2 relation(R,F,T,L,U).
```

Listagem 1.1: Factos da ontologia em *Prolog*

```
1 entity(comprador).
2 entity(endereco).
3 relation(enderecos,comprador,endereco,0,n).
```

Listagem 1.2: Exemplos de Factos da ontologia em *Prolog*

Uma interrogação corresponde à verificação da existência de um padrão de relacionamento entre classes. Neste contexto, um padrão de relacionamento retrata um conjunto de relações específicas existentes entre uma classe *standard* e as restantes, as quais tendem a ser replicadas em modelações de aplicações pertencentes ao mesmo domínio. Por exemplo, a *Listagem* 1.3 descreve o padrão da classe *standard* `Address` do domínio `eCommerce`. É definido que uma classe `U` (Utilizador) possui endereços e que uma classe `O` (Encomenda) tem pelo menos um endereço, sendo que `A` (Endereço) é uma entidade da ontologia.

```
1 ecommerce_address_pattern(A):-
2   entity(A),                    % Address is an entity
3   relation(_R,U,A,_M,_N),       % User has Address
4   has_at_least(O,A,1),          % Order has at least 1 Address
5   different(A,U), different(A,O), different(O,U).
```

Listagem 1.3: Exemplo de um padrão de relacionamento em *Prolog*

Uma vez que diferentes padrões podem coincidir entre eles, é provavél que para cada classe *standard* sejam identificadas mais do que uma entidade válida. Desta forma é necessário um mecanismo de selecção da solução mais provável de cada lista de resultados. É assim iniciada a segunda etapa.

Uma classe *standard* possui um dicionário (um documento CSV) que agrega um conjunto de sinónimos com a respetiva probabilidade de corresponderem ao nome da classe em questão. O nome de uma entidade válida é pesquisado no dicionário, sendo comparado a cada um dos registos. A percentagem de acerto é ajustada conforme o grau de semelhança com o sinónimo e a respetiva probabilidade encontrada no documento. A entidade com maior probabilidade é definida como solução da classe *standard*. Uma análise semelhante é efetuada para os atributos *standard*, tendo desta vez em conta os seus tipos respetivos.

4.2 Mapa de Conteúdo e Navegação

Nesta abordagem é utilizado um modelo auxiliar, o mapa de navegação e conteúdo. Este modelo, sob a forma de diagrama, permite representar não só as páginas, porções de páginas (parciais) e os respetivos conteúdos, bem como a navegação entre os elementos. Este diagrama unifica num modelo toda a informação necessária para a geração da IU de um dado domínio de aplicação. A representação deste mapa é baseada num diagrama de estados em UML já que este modelo permite enunciar os diferentes estados pelo qual um objeto pode transitar durante a execução de um sistema. Esta perspetiva pode ser comparada às transições entre páginas ou mudanças de conteúdo durante a execução de uma

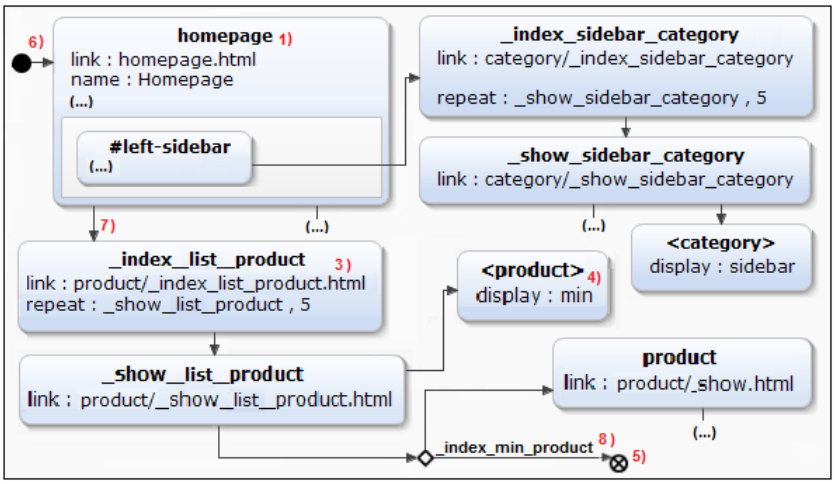


Figura 3: Exemplo demonstrativo do Mapa de navegação e conteúdo (fragmento)

aplicação baseada em *browser*. Devido à necessidade de especificar igualmente o conteúdo, foi necessário adaptar a interpretação de alguns elementos:

Estado Representa uma página, parcial, secção de página ou modo de representação de uma entidade. Os tipos associados a esta componente são distinguidos pelos identificadores ou pela sua localização no diagrama. Se o estado estiver numa região de outro estado, corresponde a uma secção da página (ver 2) da Figura 3). Os identificadores dos parciais são iniciados pelo caráter `'_'` (ver 3) da Figura 3) e os identificadores dos modos de representação de entidades são delimitados por `'<'` e `'>'` (ver 4) da Figura 3). Os restantes estados identificam páginas (ver 1) da Figura 3). Os estados possuem atributos para representar informação auxiliar.

Estado final Indica a saída de uma transição por uma condição específica (ver 5) da Figura 3).

Estado inicial Indica a página inicial da interface (ver 6) da Figura 3).

Transição Indica uma transição, substituição ou composição, dependendo do elemento de destino (ver 7) da Figura 3). Uma transição para uma página representa uma mudança de página. Uma transição para um parcial representa a substituição do conteúdo da secção de origem. Finalmente, uma transição para um modo de representação equivale a uma composição de conteúdo.

Condição Enuncia os identificadores relacionados com uma transição (ver 8) da Figura 3).

Um exemplo é apresentado na Figura 3. O mapa descreve que o sistema terá uma página inicial `homepage`. Esta página tem uma secção `#sidebar` que contém o parcial `_index_sidebar_category`, composto por uma repetição de `_show_sidebar_category`, o qual inclui o modo de representação `sidebar` da classe `Category`. Esta combinação é frequentemente usada para representar uma

lista de um parcial ou de uma entidade segundo um dado modo. A página `homepage` é composta por uma lista de `Produto` em `min`, sendo que cada elemento reencaminha para a página `product` caso o estado não seja `_index_min_product`.

4.3 Geração das componentes intermédias da IU

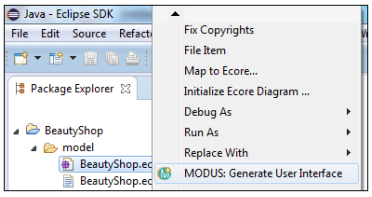
Nesta abordagem é estipulada que uma página é composta por um *layout*, que irá definir a sua estrutura, e um conjunto diferentes entidades, que irão definir o seu conteúdo. Para esse efeito os modos de representação definem, para a tecnologia da FUI, as diferentes formas de representar uma dada entidade na aplicação. Exemplos seriam uma representação completa, em miniatura, ou *hyperlink* de uma classe. Se não estiver associado a uma entidade, um modo de representação descreve um conteúdo independente da modelação, como por exemplo um *slideshow* de uma página. Por sua vez um *layout* identifica, na tecnologia da FUI, as secções que definem a página. Nomeadamente, explicita uma combinação de secções, sendo estas por exemplo *headers*, *footers*, entre outros.

O processo de criação destas componentes é executado pelo *parsing* de *templates* fornecidos pela ferramenta ou, se assim o pretender, pelo utilizador. Durante a geração, o *template* é adaptado ao *input* fornecido no programa bem como às respetivas alterações efetuadas pelo utilizador. Os *templates* são expressos em HTML, contendo atributos específicos para apoiar a criação do *output*, indicando as ações a executar, elementos da modelação, interface ou mapa de navegação. No entanto, durante a etapa final são removidos de forma a não poluírem a FUI. Os *templates* podem ser modificados pelo utilizador em *runtime*, personalizando assim as componentes intermédias que irão compor a FUI.

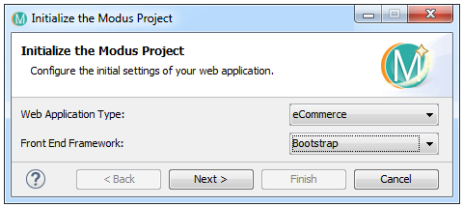
Uma criação à base de *templates* permite estabelecer um mecanismo de geração independente do resultado. A adição, remoção ou manipulação dos *templates* não influencia a implementação do processo. Esta metodologia adequa-se à área das interfaces de utilizador, em que conceitos de estética são frequentemente alterados, sofrendo uma evolução constante. A variedade de *templates* e possivéis combinações permite aumentar a flexibilidade da IU criada. Por sua vez, a importação de *templates* incrementa a diversidade de soluções geradas.

5 O *Plugin* MODUS

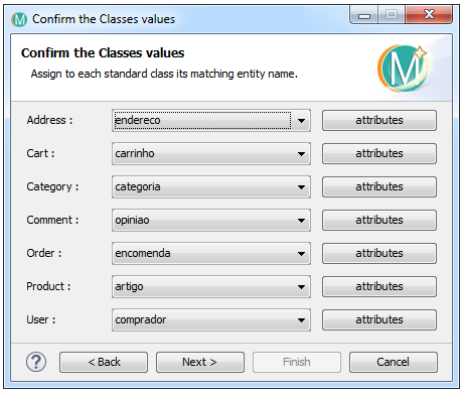
A abordagem proposta neste documento é sustentada pela ferramenta *MODUS*. Em termos de implementação, o protótipo da ferramenta é um *plugin* do *Integrated Development Environment* (IDE) *Eclipse* [31], sendo desenvolvido em *Java*. Por ser uma extensão, beneficia dos recursos disponibilizados pelo IDE, nomeadamente dos editores gráficos e da marcação de documentos para validação. Esta integração adiciona uma nova funcionalidade ao IDE, evitando que o utilizador instale um software adicional.



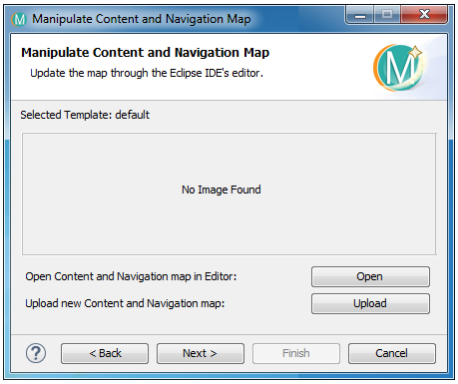
(a) Arranque do *plugin MODUS*



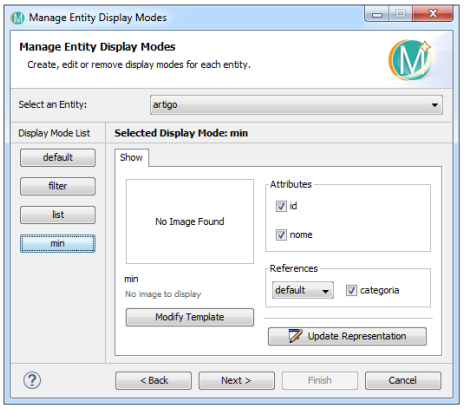
(b) *Input* do Utilizador



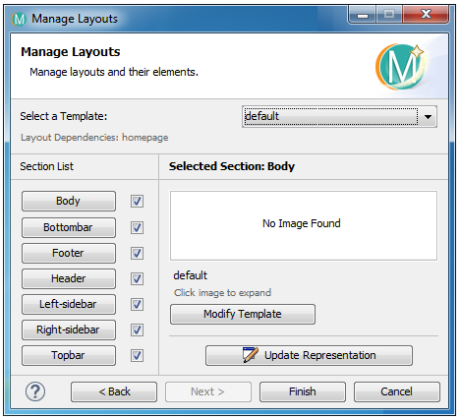
(c) Associação das classes *standard*



(d) Mapa de conteúdo e navegação



(e) Modos de representação das entidades



(f) *Layouts* das páginas

Figura 4: Interface do *plugin MODUS*

5.1 *Input* do Utilizador

Para iniciar a execução do *plugin*, é necessário importar o diagrama de classes no IDE. Uma vez carregado, pode-se selecionar, no menu de contexto associado ao formato *ecore*, a opção de gerar a IU pelo *MODUS*. Para efeitos de demonstração considera-se o modelo apresentado na Figura 2 (ver Figura 4a). Após selecionar a opção, o ficheiro é validado. Caso sejam detetados erros, é aberto, no IDE, o documento com as respetivas marcações.

A Figura 4b expõe o menu de configuração da geração, sendo o primeiro contacto com a interface do *plugin*. O menu possibilita a seleção do domínio de aplicação e das especificações da interface (as quais, no estado atual do protótipo, encontram-se incompletas). Os botões de *Next*, *Back*, *Cancel* e *Finish* permitem, respetivamente, continuar, retroceder, cancelar e terminar o processo de geração. O botão de ajuda disponibiliza alguma informação sobre a etapa atual.

5.2 Associação das Classes *standard*

A Figura 4c apresenta a interface dedicada à associação das entidades as respetivas classes *standard*. O resultado calculado é exibido, podendo ser modificado pelo utilizador. Os botões *attributes* abrem uma janela semelhante dedicada aos atributos da entidade. Pelo momento, a gama de atributos *standard* conhecidos pela ferramenta é reduzida.

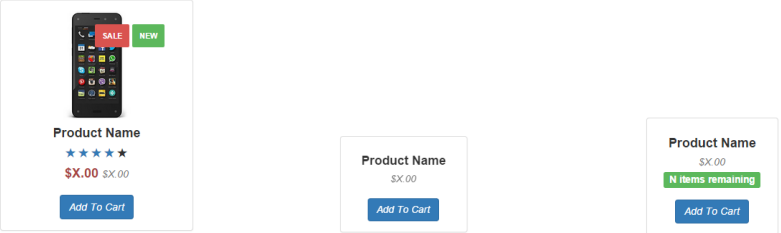
5.3 Mapa de Conteúdo e Navegação

A Figura 4d apresenta a interface dedicada ao mapa de navegação e conteúdo (ver exemplo da Figura 3). Caso pretenda, o utilizador pode alterar o mapa através do botão *Update*, o qual desencadeia a abertura do diagrama no editor gráfico disponibilizado pelo IDE. O utilizador pode igualmente importar outro modelo da lista de mapas da ferramenta, através do botão *Upload*.

5.4 Modos de Representação das Entidades

A Figura 4e expõe a interface dedicada aos modos de representação das entidades. O *drop down* situado no topo (*Select an Entity*) permite selecionar uma entidade. Esta escolha desencadeia a listagem, no lado esquerdo da interface (*Display Mode List*), dos modos de representação associados à entidade. A seleção de um dos modos, preenche o lado direito da interface com a respetiva informação. Os *tabs* permitem aceder aos tipos de apresentação disponíveis para cada modo: exibição (*Show*), que exibe a entidade na aplicação segundo um dado modo, e formulário (*Form*), que apresenta o formulário na aplicação do modo em questão. No exemplo da Figura 4e o modo `min` é apenas composto pelo tipo *Show*. Para cada alternativa é referido o *template* associado e as componentes da entidade permitidas pelo mesmo.

O botão *Modify Template* permite associar um novo *template*, quer seja selecionado a partir de uma lista pré-definida, quer seja carregado pelo utilizador. O



(a) *Template* original do modo `list` da entidade `artigo` (ver Figura 2)

(b) *Template* ajustado do modo `list` da entidade `artigo` (ver Figura 2)

(c) *Template* modificado do modo `list` da entidade `artigo` (ver Figura 2)

Figura 5: Evolução de um modo de representação na ferramenta

template é atualizado conforme a respetiva modelação antes de ser apresentado na ferramenta (ver Figura 5a). A ter em conta que no caso particular do *template* da Figura 5b há uma restrição do conteúdo a um grupo de atributos específicos. Por sua vez, O botão *Update Representation* permite editar o modo no editor HTML do IDE, tal como demonstrado na Figura 5c.

5.5 *Layouts* das *Views*

A Figura 4f apresenta a interface dedicada aos *layouts* das páginas. Esta janela aproxima-se à interface dos modos de representação de entidades, sendo a sua interação com o utilizador semelhante. O *drop down* (*Select a Layout*) permite selecionar um layout. A listagem à esquerda enumera as secções de um *layout*, sendo que cada *checkbox* define a existência da respetiva secção no *layout*. À direita apresentam-se os dados da secção selecionada. Os *templates* são igualmente ajustados conforme as alterações efetuadas no *plugin*.

5.6 Geração da interface

Após terem sido efetuados os ajustes às componentes intermédias da interface, é possível iniciar a criação da FUI. A interface final será composta por elementos utilizados no código da lógica de negócio: os parciais (ver exemplo da Figura 6) e os *layouts* (ver exemplo da Figura 7) das páginas da aplicação. Após a geração desses elementos, a FUI será acompanhada por um conjunto de páginas estáticas que irão simular a interface por completo (ver exemplo da Figura 8) . Estas páginas são elaboradas pela composição dos *layouts* com os *parciais*, conforme a sua definição no mapa de navegação e conteúdo.

Note-se que, a interface apresentada nas Figuras 7, 6 e 8, representa uma versão simples da IU gerada automaticamente. O objetivo é mostrar que é viável gerar as interfaces com conteúdos relevantes para a aplicação. No entanto, através do recurso a *front-end frameworks* apropriadas, está já a ser desenvolvido suporte à personalização avançada da interface.

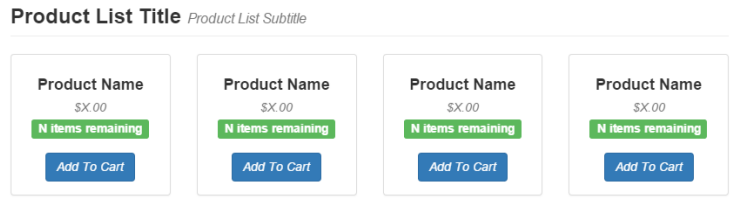


Figura 6: Parcial `_index_list_product` (ver Figura 2)

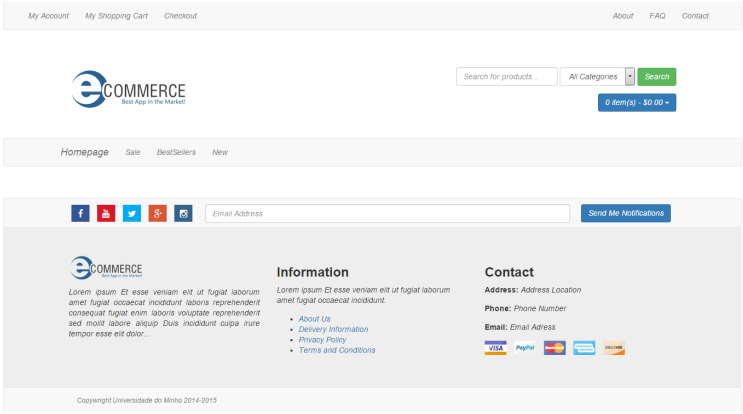


Figura 7: *Layout* `default` (ver Figura 2)

6 Conclusões e Trabalho Futuro

Este documento apresentou uma abordagem baseada em MDE e MBUID para a geração de interfaces de utilizador com um alto nível de automação. Na solução proposta o diagrama de classes e o domínio da aplicação são elementos chave. O diagrama de classes enuncia todas as entidades que irão intervir na IU. Por sua vez, o domínio é a base de diversas assunções sobre a interface, substituindo-se a necessidade de a modelar detalhadamente.

A implementação dos modos de representação de entidades e dos *layouts* é baseada na separação do conteúdo e estilo da interface de aplicações web baseadas em *browser*. Desta forma, os *templates* permitem definir detalhes estéticos da IU, contribuindo para a elaboração de interfaces complexas e apelativas. Aincorporação de *front end frameworks* e, posteriormente de restantes recursos de estética, permite tirar partido dessa contribuição, possibilitando igualmente aumentar a compatibilidade entre dispositivos/plataformas e melhorar a aparência da própria interface.

Para suportar a abordagem está a ser desenvolvido um *plugin MODUS* para o *Eclipse*. Um protótipo foi elaborado, implementando por completo as três etapas intermédias para a conceção da interface do utilizador. A geração da FUI produz parciais, *layouts* das páginas e um conjunto de páginas estáticas, pos-

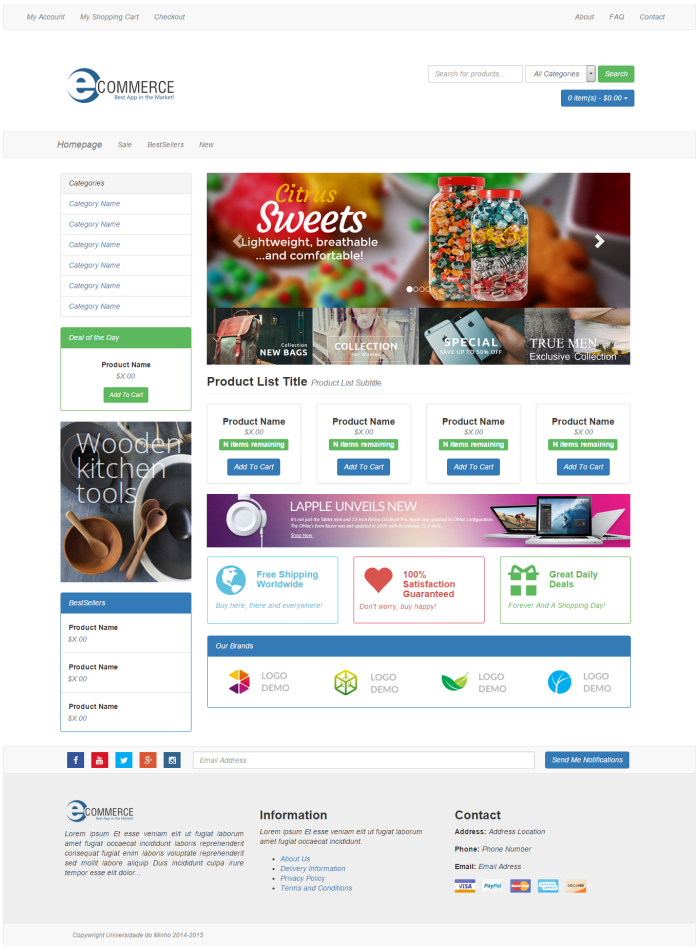


Figura 8: Página `homepage` (ver Figura 2)

suindo um alto nível de automatização. Constata-se que o grau de automação é influenciado pela intervenção do utilizador, o qual permite no entanto aperfeiçoar as componentes que irão compor a interface, garantindo-se melhores resultados.

Como trabalho futuro, tenciona-se adicionar novas possibilidades de configuração da interface, nomeadamente a seleção de temas visuais e paletas de cores. Para além disso pretende-se completar o conjunto de atributos *standard* reconhecidos pela ferramenta, de forma a suportarem um vasto leque *templates* complexos, os quais deverão igualmente ser desenvolvidos.

A validação da abordagem incluirá duas componentes distintas. Por um lado, será necessário comparar a implementação de interfaces utilizando a ferramenta com o desenvolvimento seguindo abordagens mais manuais, por forma a avaliar precisamente qual a poupança em termos de esforço de programação. Isso po-

derá ser realizado, avaliando quer o tempo gasto, quer o código produzido nos projetos em cada uma das condições. Por outro lado, será necessário comparar a qualidade das interfaces produzidas, a fim de avaliar qual o impacto do processo de automatização na qualidade final das interfaces

Referências

1. Brad A. Myers, Scott E. Hudson, and Randy F. Pausch, "Past, present, and future of user interface software tools", ACM Trans. Comput.-Hum. Interact., 7(1):3–28, 2000.
2. Gerrit Meixner, Fabio Paternò and Jean Vanderdonckt, "Past, present, and future of model-based user interface development", i-com, 10(3):2–11, 2011.
3. Brad A. Myers and Mary Beth Rosson, "Survey on user interface programming", Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '92, 195–202, ACM, 1992.
4. Sanjay Mittal, Clive L. Dym and Mahesh Morjaria, "Pride: An expert system for the design of paper handling systems", Computer, 19(7):102–114, 1986.
5. D. G. Bobrow, S. Mittal and M. J. Stefik, "Expert systems: Perils and promise", Commun. ACM, 29(9):880–894, 1986.
6. Richard Kennard and Robert Steele, "Application of software mining to automatic user interface generation", New Trends in Software Methodologies, Tools and Techniques - Proceedings of the Seventh SoMeT, 244-254, 2008.
7. Brad A. Myers, "State of the art in user interface software tools", Technical report, Carnegie Mellon University, USA, 1992.
8. Brad A. Myers, "User interface software tools", Technical report, Carnegie Mellon University, USA, 1994.
9. Pedro J. Molina, "A review to model-based user interface development technology", MBUI, Proceedings of the First International Workshop on Making model-based user interface design practical: usable and open methods and tools, Portugal, 2004.
10. Gerrit Meixner and Gaelle Calvary "Introduction to model-based user interfaces", W3C, 2014, http://www.w3.org/TR/2014/NOTE-mbui-intro-20140107/.
11. Fowler, "UML Distilled: A Brief Guide to the Standard Object Modeling Language"AddisonWesley Longman Publishing Co., Inc., 3 edition, USA, 2003.
12. A.M. Davis, "Operational Prototyping: A New Development Approach", IE Software, 9(5):70-78, 1992.
13. Thomas Stahl, Markus Voelter, and Krzysztof Czarnecki, "Model-Driven Software Development: Technology, Engineering, Management", John Wiley & Sons, 2006.
14. Jorg Rech and Christian Bunse, "Model-Driven Software Development: Integrating Quality Assurance", Information Science Reference - Imprint of: IGI Publishing, Hershey, 2008.
15. Egbert Schlungbaum, "Model-based user interface software tools current state of declarative models", Technical report, Graphics, Visualization and Usability Centre, Georgia Institute of Technology, GVU Tech Report, 1996.
16. P. Kidwell, "The mythical man-month: Essays on software engineering", IEEE Ann. Hist. Comput., 18(4):71–, 1996.
17. B. Hailpern and P. Tarr, "Model-driven development: The good, the bad, and the ugly", IBM Syst. J., 45(3):451–461, 2006.
18. Paulo Pinheiro Da Silva, "User interface declarative models and development environments: A survey", Proceedings of the 7th International Conference on Design,Specification, and Verification of Interactive Systems, 207–226, 2001.

19. Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt "A unifying reference framework for multi-target user interfaces", Interacting With Computers, 15:289–308, 2003.
20. Jean Vanderdonckt "A mda-compliant environment for developing user interfaces of information systems", Proc. of 17th Conf. on Advanced Information Systems Engineering, CAiSE'05 16–31, Springer-Verlag, 2005.
21. Jean M. Vanderdonckt and François Bodart, "Encapsulating knowledge for intelligent automatic interaction objects selection", In Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems, CHI'93, 424–429, ACM, 1993.
22. Dennis J. M. J. de Baar, James D. Foley, and Kevin E. Mullet, "Coupling application design and user interface design", In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI'92, 259–266, ACM, 1992.
23. Pedro Szekely, Ping Luo, and Robert Neches, "Beyond interface builders: Model-based interface tools", In Proceedings of the INTERCHI'93 Conference on Human Factors in Computing Systems, INTERCHI'93, 383–390, 1993.
24. Angel R. Puerta, Henrik Eriksson, John H. Gennari, and Mark A. Musen, "Beyond data models for automated user interface generation", In Proceedings of the Conference on People and Computers IX, HCI '94, 353–366, 1994.
25. Silvia Berti, Francesco Correani, Giulio Mori, Fabio Paternò, and Carmen Santoro, "Teresa: A transformation-based environment for designing and developing multi-device interfaces", In CHI '04 Extended Abstracts on Human Factors in Computing Systems, CHI EA '04, 793–794, ACM, 2004.
26. A.I. Sampaio and J.C. Campos, "Towards a Framework for Adaptive Web Applications", In C. Stephanidis, HCI International 2014 - Posters' Extended Abstracts, Part I, V.434 of Communications in Computer and Information Science, 240-245, Springer, 2014.
27. Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, Víctor López-Jaquero, "Usixml: A language supporting multi-path development of user interfaces", Engineering Human Computer Interaction and Interactive Systems, V. 3425 of Lecture Notes in Computer Science, 200–220, 2005.
28. IT Resources, Outsystems, http://www.outsystems.com/itresources/, 2011.
29. Integranova Software Solutions, Integranova, http://www.integranova.com/, 2005.
30. Frank Budinsky, Dave Steinberg, Ed Merks, Ray Ellersick, Timothy J. Grose, "Eclipse Modeling Framework : A Developer's Guide", Addison-Wesley Professional, 14–20, 2003.
31. The Eclipse Foundation, Eclipse, https://eclipse.org/, 2001.
32. Rui Couto, António N. Ribeiro, and José C. Campos, "Mapit: A model based pattern recovery tool", V.7706 of Lecture Notes in Computer Science, 19–37, 2013.

SpreadsheetDoc: An Excel Add-in for Documenting Spreadsheets

Diogo Canteiro and Jácome Cunha

Universidade Nova de Lisboa, Portugal
d.canteiro@campus.fct.unl.pt
jacome@fct.unl.pt

Abstract. Documentation is an important artefact of any software product. This is also the case for spreadsheets were, even considering an industrial setting, only 30% have some kind of documentation. This makes their usage and maintenance very difficult.
In this paper we describe a tool, SpreadsheetDoc, that allows users to document spreadsheets in a structured way, allowing them to describe different parts of spreadsheets. For instance, for (future) spreadsheet users, it is possible to describe input and output cells, and for (future) developers, it is possible to describe computation, that is, formulas: their arguments, their internal computations, and their outputs.

Keywords: Spreadsheet, Documentation, Tool, Excel, Add-in

1 Introduction

Nowadays, there are a huge number of people using spreadsheets. In fact, spreadsheet systems are the most used programming system [13], specially by non-professional programmers, the so-called end users. As in other programming languages/environments, it is quite common to find spreadsheets with errors. In fact, the error rate within spreadsheets can be up to 90% [15]. The European Spreadsheet Risk Interest Group (EuSpRIG)[1] regularly updates their web site with new stories reporting the loses (economical, brand recognition, etc.) caused by errors in spreadsheets to companies and other entities.

Many reasons exist for this scenario: the lack of abstraction, of a testing methodology, or a (very) weak type system. Some errors can also be explained by the lack or poor documentation [18]. Indeed, in many of the cases reported by EuSpRIG, the lack of or bad documentation is mentioned. Moreover, software tends to lose some of its efficiency when no proper documentation is available [7]. Without the proper documentation users and developers have more difficulties in understanding, using, and updating the software. The same happens for spreadsheets. In a recent study in a financial institution, researchers found that 70% of users that receive spreadsheets from colleagues have difficulties understanding

[1] http://www.eusprig.org/

them [11]. This transferring scenario is quite common as 85% of the study participants reported doing so. The same authors report that spreadsheet users browse them for hours trying to understand them since only 1/3 are documented [10].

Unfortunately, spreadsheet systems do not have a proper form to document their programs. In modern spreadsheet systems it is possible to add general notes to a cell, but that is a very unstructured way of doing documentation, when compared to what tools like JavaDoc allow. This can be compared as to write ad-hoc comments in a textual language. This makes it quite hard for spreadsheet developers to actually document their spreadsheets. For instance, in [9] the authors analysed more than 15.000 spreadsheets available in the Enron Email Archive [12], containing the emails from the Enron corporation. They found that some spreadsheet documentation was in the emails themselves, instead of being in the spreadsheets. This shows that there is the need to document, but not the proper means.

Users tend to workaround this situation documenting their spreadsheets as possible. Some write the documentation on a separate worksheet and reference to it informing that such worksheet is the documentation of the spreadsheet. In this case, it is not possible to see the documentation and the corresponding document artefacts at the same time, as one can do using for instance JavaDoc, making it difficult to relate the documentation with the actual spreadsheet content. Others write the documentation on the worksheet with the content, close by the cells they want to describe. However, in these cases users are inserting extra cells in the spreadsheet, which are not part of the program, making it more complex.

These kinds of documentation make users question its use. Although it is important to document software, it cannot be done in any way. It is important to write and organize it in such a way that the target readers will get what they want. A good documentation will increase the users' efficiency and effectiveness, and thus, their productivity [7, 16]. Indeed, JavaDoc, for the Java programming language, is a good example of a successful way of documenting software.

In this paper we present a tool, SpreadsheetDoc, described in Section 3, to guide spreadsheet developers to write proper documentation. In this case, two kinds of documentation should be written, as suggested in [20]: i) documentation for end users, and ii) documentation for developers. Notice that both these types of documentation should be written by the spreadsheet developers, but part is intended to be read by end users and part by developers.

Spreadsheet end users are the ones interested in executing the spreadsheet to compute the results they want. Thus, they are mostly interested in understanding which cells they should fill in to feed the program, that is, the input, and were they can find the results, that is, the output. So, spreadsheet developers should mark and document all the input and output cells, and write documentation for users, and not for developers, that is, simple and straightforward documentation. The purpose of the spreadsheet file, and of each worksheet also fits in this category so users can find the spreadsheet and corresponding worksheet they need. We describe in detail how to do this in Section 4.

On the other hand, maintainers and future developers of the spreadsheet must have more technical information about the computations performed. Thus, cells containing complex formulas should also be documented. In this case, the documentation should be technical so others can later correct or evolve the spreadsheet. We discuss this is Section 5

With our tool we also allow to document a particular column, row, or range of cells (for instance, a table in a spreadsheet, that is, a range separated of the remaining cells by empty columns and rows).

Users can then read the documentation within the spreadsheet itself, in the context of the part of the spreadsheet they are using, or read the complete documentation in a web page, which is generated by our tool in a similar way as JavaDoc. To generate the web page we first create an XML file where the complete documentation is saved. Although we decided to present to users a web page, this intermediate format makes it possible to present documentation in a different way. Moreover, it also allows to import documentation produced by other systems, as long as the correct XML file is available. This allows to publish inside a corporation all the documentation of all spreadsheets, making it easier for collaborators to find one that already does what they need. Moreover, since spreadsheets can reference other spreadsheets, the navigation to the corresponding documentation is straightforward, as links connect them.

In Section 2 we present a running example, in Section 6 we discuss related work, and in Section 7 we present our conclusions and future work.

2 Motivational Example

2.1 Definitions

Before we introduce our motivational example, we will define a few concepts from the spreadsheet realm.

Workbook/Spreadsheet A workbook is a spreadsheet file. The term spreadsheet is often used to refer to a workbook, when in fact it refers to the computer program, such as Excel. We will use these terms interchangeably.

Worksheet A worksheet, or simply sheet, is a single page of a workbook, that is, one of the tabs that can be found at the bottom of the spreadsheet (in most spreadsheet systems).

Cell A cell is a rectangular box in a worksheet, that is, the intersection point of a vertical line (column) and a horizontal line (row). Its name is the concatenation of its coordinates: a letter for the column and a number for the row. It also has content, which can be plain values (for instance, `4` or `Bid`), or formulas (for instance, `=SUM(A1:A3)`).

Row Refers to all the cells contained in a horizontal line (given by a number).

Column Refers to all the cells contained in a vertical line (given by a letter).

Range A range is a group of cells in a worksheet that form a rectangular area.

Input Cell A cell referenced by others, but not referencing any other cell.

Output Cell A cell that references other cells, but its not referenced by others.

2.2 Example

We now describe a spreadsheet which we will use as a running example. This spreadsheet, shown in Figure 1, was introduced in a book describing how to create spreadsheets [14].

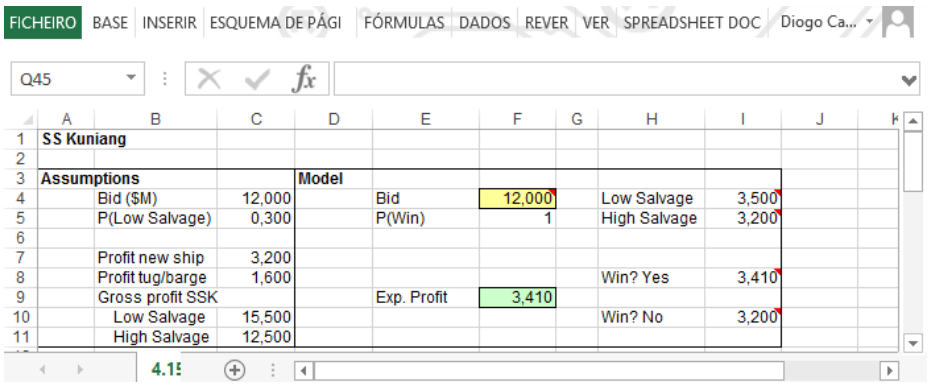


Fig. 1. A spreadsheet to calculate the winning probabilities of an auction.

This spreadsheet calculates the probability of winning an auction, according to a set of assumptions. Although this spreadsheet is well organized and rather small, it is already difficult to understand. In Figure 2 we show the same spreadsheet, but now with the formulas visible.

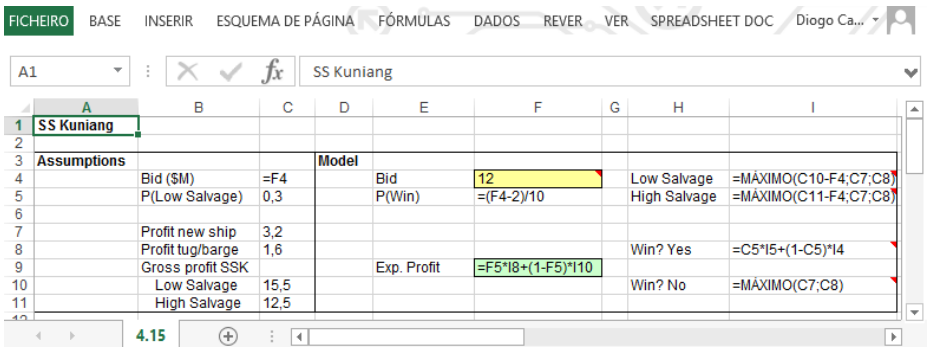


Fig. 2. Spreadsheet of Figure 1 with formulas visible.

In fact, and since this is a well designed spreadsheet, some cells even have comments on them (denoted by the small triangle in the top right corner of the corresponding cells). We list next the comments from the spreadsheet:

F4 Decision: Bid (in $million)

I4 Net value if the salvage value is low.

I5 Net value if the salvage value is high.

I8 Net value if the bid is successful.

I9 Net value if the bid is unsuccessful.

In the book, one can read some more details about this spreadsheet and corresponding computations. What we envision is a system where one can describe the different parts of the spreadsheet, but in a systematic approach, and in the corresponding context.

For instance, cell F5 calculates the probability of winning the auction. The formula present in the book is `P(Win) = (Bid - 2)/10, for 2` $\leq$ `Bid` $\leq$ `12`. This formula is more direct than the one presented in the spreadsheet, and the range of `Bid` (F4) is now clear. This should be part of that cell's documentation.

Using our approach, to document such formula, the user would click on the button to describe cell ("Cell"), under the group "Content Documentation", and the wizard shown in Figure 3 would appear:

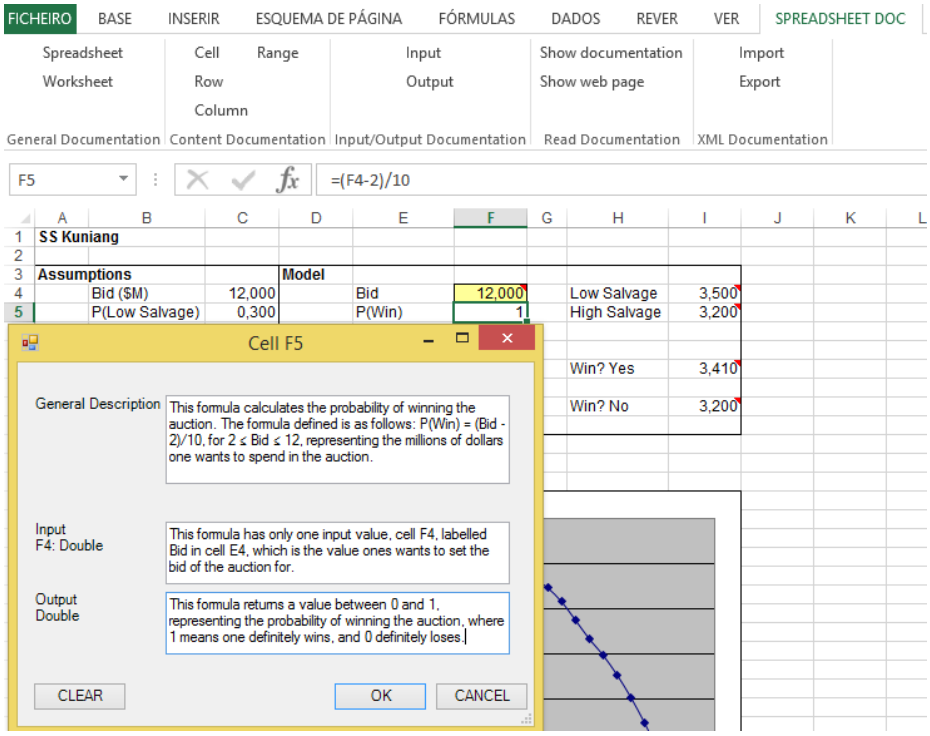


Fig. 3. Dialogue to document a formula cell.

Since we are documenting a cell, as in any other programming language, the developer should describe the computation, the input, and the output. The first text box allows the user to write a general description of the formula. This is similar to what a Java programmer starts to write when documenting a method. Next, the user can describe the argument of such formula. In this case, the input is cell F4, which is the label of the text box. Moreover, the tool also shows the type of the inputs, in this case, double. This may also help the user to detect incorrect usage of cells. Finally, the user can describe the output of the formula, again annotated with the corresponding type.

In the following sections we will describe in detail all the features of our approach, including how to describe input, output, and computations.

3 The SpreadsheetDoc Add-in

We have developed our tool, SpreadsheetDoc, as an add-in for Excel 2013. To develop our framework we used the programming language C#, version 4.0, and Visual Studio Ultimate 2013. To use our SpreadsheetDoc it is necessary to install it as an add-in within Excel. It will appear as a new ribbon with its name and when the user clicks on it he/she will have available all its functionalities. Each of these functionalities is implemented as a method. Thus, the code structure is simple allowing to easily add new methods/functionalities. Since each functionality can be used to add new documentation or to update existent one, each method must verify if some documentation already exists for the selected spreadsheet part (worksheet, cell, etc.). If no documentation exists, then a new form must be created. Otherwise, the form is loaded with the existing documentation.

Our framework is structured in three parts. The first part is where the user writes, reads, and updates all the documentation. This is done using the corresponding buttons listed in the ribbon and described in Sections 4 and 5.

The second part is the possibility of importing and exporting XML files with the documentation. Such file can be used in different ways. For once, it is used by the tool itself to create a web page where the user can read the spreadsheet documentation, possibly with links to documentation of other spreadsheets, if they are referenced.

When exported, this XML file can also be used by other tools as they wish. For instance, it can be used by other Excel add-ins to show the documentation in a different way, or by add-ins for other spreadsheet system such as LibreOffice or OpenOffice so they can open Excel spreadsheets, but also their documentation.

It is also possible to read an XML file to import documentation written in other tools. This makes it easier to exchange spreadsheet documentation. For instance, it allows the user to import a new version of the documentation the developer may have written. It may also be used to import documentation written for that spreadsheet, but using another spreadsheet software, like OpenOffice.

The third part is where the user reads the documentation on a web page. This web page is generated based on the XML file already created. Such web page can potentially be consulted by other people. For instance, inside an organization

there can be a server with all the web pages of all spreadsheets available, and users can search for some spreadsheet implementing a functionality they need.

Figure 4 illustrates the potential interactions users can have with the environment SpreadsheetDoc creates.

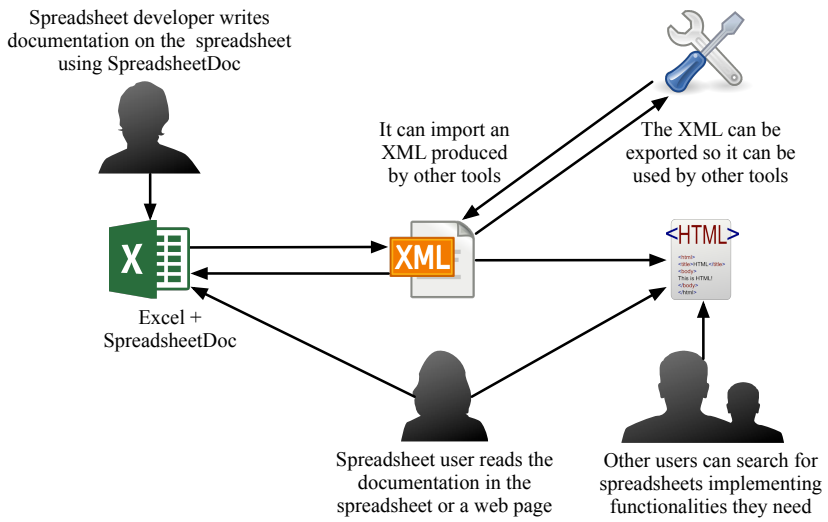


Fig. 4. The possible interactions with the SpreadsheetDoc environment.

The tool can be downloaded in the following web page:
https://bitbucket.org/spreadsheetdoc/spreadsheetdoc/.

4 Documentation for Spreadsheet Users

In this section we will describe the SpreadsheetDoc features that allow to write documentation more directed for spreadsheet end users. An end user is the person who uses the spreadsheet after it has been fully developed. Indeed, they probably do not even completely understand how it computes the results. Thus the documentation should be easy to understand. Spreadsheet end users only intend to input values and read the computed output. They can be seen as any other software end users. So, spreadsheet developers should write documentation using the next features focusing on end users. Notice that all the documentation should be written by developers. However, in some cases it is intended to be read by end user and in other cases by developers. Indeed, each of the features we present can be used to write and to read documentation, but should be used by developers to write and by users to read.

SpreadsheetDoc is composed of five different groups of functionalities: General Documentation, Content Documentation, Input/Output Documentation,

Read Documentation, and XML documentation. In the following we describe each functionality of each group.

4.1 Documenting a Spreadsheet Program

The first functionality we introduce, part of the General Documentation group, is the one to document an entire spreadsheet document.

In general an organization makes use of many spreadsheets. For instance, for the oil company Enron there were more than 15.000 spreadsheets exchanged in their emails [9]. Thus, it is important to document each spreadsheet file, so users can know each one and possibly reuse them.

So, the first documentation the user should write is about the spreadsheet itself. The "Spreadsheet" button in our add-in opens a dialogue box with a text box inside. The user can then write the spreadsheet's general purpose. In this dialogue three buttons are shown: `clear`, `ok`, and `cancel`. The `clear` button, as the name suggests, clears the text box. The `ok` button saves the dialogue box state. Finally, `cancel` drops all changes inside the dialogue box. Figure 5 shows the wizard for our running example, and the description we added.

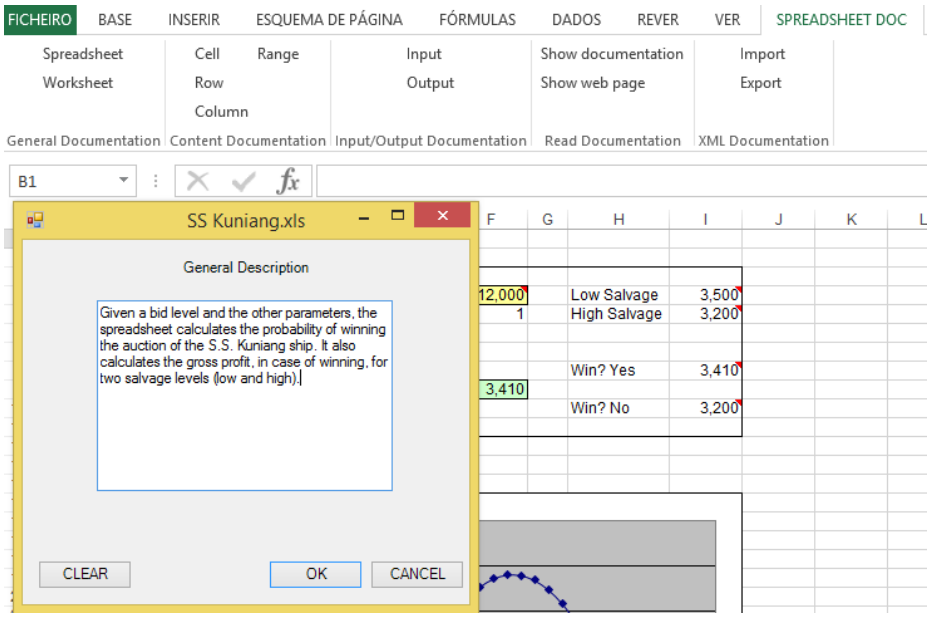


Fig. 5. The "Spreadsheet" button wizard to document a spreadsheet file.

After writing and saving the documentation, if the user clicks again in the button, the same wizard will be shown, but this time showing the recorded text. This behaviour is the same for all the functionalities.

4.2 Documenting Each Worksheet

A spreadsheet can have dozens, even hundreds of worksheets. For the EUSES spreadsheet corpus the biggest spreadsheet has 106 worksheets [8] and for the Enron 175 were found in a single spreadsheet file [9]. Thus, it is quite important to document each of these worksheets, otherwise it becomes impossible to know what each one is doing.

The "Worksheet" button, from the General Documentation group, has a structure similar to the previous button, but in this case users should document the behaviour of the worksheet and not of the spreadsheet. Inside each worksheet the user should click this button and write the corresponding documentation. It will be associated with the worksheet the user is in.

So, the documentation generated by the General Documentation group has the goal of giving users a better understanding of the spreadsheet behaviour, helping them understanding what was it developed for (Spreadsheet button) and how each of its pieces work (Worksheet button). The wizard shown is similar to the one in Figure 5, and thus we do not show it.

4.3 Documenting a Cell

The finest grain in a spreadsheet structure is a cell. Although in most cases it is not necessary to document each cell individually, some of them must be documented so one can understand how the spreadsheet works. Since for now we are focused on user documentation, the description about documentation of cells with formula is left for Section 5.

The "Cell" functionality, from the group Content Documentation, can be used to document each and every cell. If the cell is a plain value, then the wizard shown is similar to the one in Figure 5 (thus we do not show it). The documentation writer can then describe the cell content. If the content is a formula, then the description must be more technical, so it can be updated by other developers. We discuss this in Section 5.

4.4 Documenting a Column

Usually, spreadsheet developers tend to organize the data by rows or columns. For now we are going to focus on columns. Depending on the spreadsheet structure, commenting a column can be useful. Indeed, a user may document an entire column describing its behaviour (for instance, by saying the column computes the average of the columns before it). This functionality can also be used when each column of the worksheet has a particular meaning. For instance, in Figure 6 one may wander what each column represents.

In fact, this spreadsheet has another worksheet containing its documentation. For instance, it clarifies that column `N (OK)` has the value `1,00` in all its cells.

The use of this functionality makes sense only if the worksheet contains a single table. If it has more than one, commenting an entire row can be confusing.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	COM_CODE	UPC	DESCRIP	SIZE	CASE	NITEM	STORE	WEEK	MOVE	QTY	PRICE	SALE	PROFIT	OK
2	653,00	1111140009	DOVE DISH LIQUID	42 OZ	9,00	2851281	100	383	16	1	2,19		33,47	1,00
3	653,00	1111140009	DOVE DISH LIQUID	42 OZ	9,00	2851281	100	384	7	1	2,19		33,47	1,00
4	653,00	1111140009	DOVE DISH LIQUID	42 OZ	9,00	2851281	100	385	15	1	2,19		33,47	1,00
5	653,00	1111140009	DOVE DISH LIQUID	42 OZ	9,00	2851281	100	386	14	1	2,19		38,49	1,00
6	653,00	1111140009	DOVE DISH LIQUID	42 OZ	9,00	2851281	100	387	14	1	2,19		43,51	1,00
7	653,00	1111140009	DOVE DISH LIQUID	42 OZ	9,00	2851281	100	388	14	1	2,19		43,51	1,00

Fig. 6. A spreadsheet representing dishwasher detergents taken from [14].

The "Row" button opens a dialogue box with a text box inside, similar to the one presented in Figure 5 (thus we omit its illustration). This button is also part of the Contend Documentation group.

4.5 Documenting a Row

In the previous sub-section we described the documentation of a column. The dual applies for rows. In some cases, spreadsheets are developed column oriented, but in some other cases, row oriented. Indeed the example shown in Figure 6 could be organized by rows instead of columns. Thus, the documentation writer must choose the adequate functionality so the user can get the most out of the documentation.

4.6 Documenting a Range

Spreadsheets are a development framework where users have freedom to do what they want. So, it is possible (and actually quite common) to have more than one table on the same worksheet. Then, tables have different objectives and it is important to understand what is the purpose of each one. So, we created a functionality where it is possible to document a selected range of cells, that is, a rectangular selection of cells.

The "Range" button opens a dialogue box with a text box inside so the user can document the selected range. Again, this is similar to what is shown in Figure 5. This button is the last of the Content Documentation group. The documentation generated by this group focus on the understanding by users of cells, rows, columns, and ranges content.

4.7 Documenting an Input Cell

The third group, Input/Output Documentation, has two functionalities: documenting input cells, and documenting output cells.

The "Input" button opens a dialogue with two text boxes inside. The first text box is in read-only mode and shows a list of all input cells and corresponding type, as given by Excel (for instance, Double). As we said, it is read-only and users cannot modify it. After the user adds a new input cell, the list is updated. The second text box allows the user to freely document the cell. A cell is added to the list after the user clicks `ok`. In such a dialogue four more buttons are sown: `clear`, `ok`, `cancel`, and `remove`. The three first buttons act as described before.

The `remove` button allows the user to remove the current input cell from the list. This is only possible if the cell is in the list already.

Note that it is possible to document a cell using the "Cell" and the "Input" functionality simultaneously. The fact that the cell is an input point is important for end users, and thus it should be documented as such. However, the spreadsheet developer may feel the need to add more technical details to such cell, which may not be of interest for end users, but only for a future developer.

4.8 Documenting an Output Cell

Output cells are where the user usually sees the results produced by the other cells. Thus, these are probably the most important cells for end users.

Similar to "Input", the "Output" button opens a dialogue with two text boxes inside, one (in read-only mode) showing a list with all the output cells (name and type) of the spreadsheet (top part of Figure 7), and a text box to describe the current selected output cell (bottom part of Figure 7). For each output cell, it is also shown its type, as given by Excel. In Figure 7 we show the wizard for this case, documenting a cell of our running example.

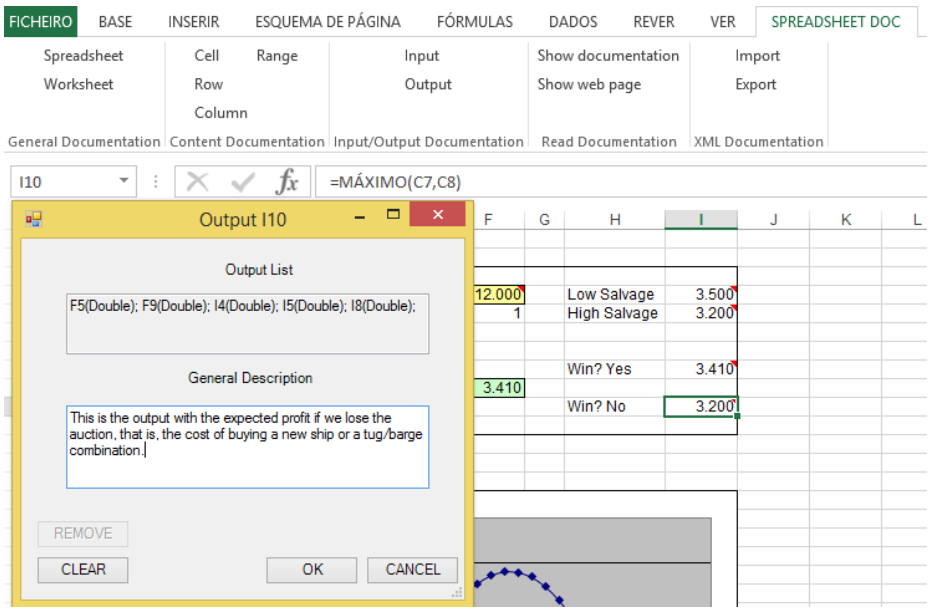


Fig. 7. Dialogue to document an output cell.

Again, notice that output cells may also be documented as cells to explain more technical details about its content (see Section 5).

4.9 Show Cell Documentation

The fourth group, Read Documentation, has two functionalities: one to read the documentation on the spreadsheet, and another to read it on a web page.

The "Show documentation" functionality opens a dialogue showing the user the selected cell documentation. This can be documentation either from the "Cell" functionality, from the "Row" functionality, or from any other content documentation. This allows the user to see the complete documentation of a cell with a single click.

This button has two possible states: enabled and disabled. If the button's state is disable it represents that there is no documentation for the selected cell. Otherwise, it is possible to click on it and read the existent documentation. Nevertheless, the user can always click on each individual functionality to see the corresponding documentation.

4.10 Show Documentation Web Page

The last user documentation functionality, is also the last one from the Read Documentation group. The "Show web page" button opens a web page showing all spreadsheet's documentation. The web page is created locally using the documentation previously written. This allows to show the documentation in a more appealing fashion. Moreover, references to cells are links that can be clicked to read the corresponding documentation. In fact, if the spreadsheet references other spreadsheets, and they have documentation, it can also be read.

5 Documentation for Developers

Spreadsheet developers are the ones designing, implementing, and documentation the spreadsheet. When they write documentation, they should know the target people that will use their spreadsheet [16]. Indeed, they should distinguish documentation for the spreadsheet users and (future) maintainers. The documentation for developers may have complex details and technical concepts. The SpreadsheetDoc feature we now describe allows developers to write documentation for cells they consider complex, for instance, cells with formulas.

Indeed a spreadsheet can have thousands of cells: for the EUSES spreadsheet corpus the biggest spreadsheet has 889.952 [8], and for the Enron 113.134 [9]. With our framework it is possible to document each cell individually (although in many cases this is not necessary). The developer must decide which ones deserve to be documented.

We have previously described the "Cell" functionality to document a particular cell, in the context of end users. Such functionality can also be used to described formula cells. The "Cell" button opens a dialogue that is contextualized with the cell content. If the cell contains a plain value (a number or a string), only a text box is presented as we presented in Section 4.3. On the other hand, if the cell content is a formula, at least three different text boxes must be filled in, as shown in Figure 3:

1) The first text box is for the user to write a small description of the selected cell. The next text boxes are used to describe the input.

2) For each input, that is, for each reference or range in the formula, a text box is presented so the user can describe such input. Each text box has a label on the left showing two possible options:

   a) If the input is a cell range, such range is shown, so the user knows which cells he/she is describing. The range type is also shown. The type of a range however must be computed by our tool as Excel does not have such information. If all the cells have the same type, then such type is presented. Otherwise, we compute the type represented in more cells and present that type, showing the remaining types and corresponding cells.

   b) If the input is a reference to a single cell, then the tool presents its name (reference) and type as given by Excel. This can be seen in Figure 3 for our running example.

3) Finally, the last text box is used to describe the output generated by the cell. Its label is the type of the cell.

This documentation process can be compared to the JavaDoc tool where users document their methods. In this case, developers document formulas. With JavaDoc the user writes a general description of the method, our first box, describes each method argument, our following boxes, and finally describes the return of the method, our last box.

6 Related Work

In [19] the authors present a system to format spreadsheet documentation. This system uses an external editor to document spreadsheets and macros to format such text. This can be compared to literate programming, where documentation and code are kept together in the same file. The objective is to easily write and update documentation for spreadsheets. However, the user must "program" the documentation itself, for instance, writing the following code to create a variable to document a formula: `@MACRO(xfor(v)=[@P(@V(f_@X(v)))])`. These variables can then be assigned to parts of the spreadsheet and used in the text documentation. However, this seems difficult to learn, specially for end users. SpreadsheetDoc on the other hand shows contextualized wizards with the necessary text inputs the documentation writer must fill in, making it more convenient for end users to document their spreadsheets.

Raymon did a study showing that a good documentation improves mainly two factors: effectiveness and efficiency of users [16]. So, our framework intends to improve these two factors. Raymon describes that users usually ask to colleagues when they do not understand the system they are working with. Others spend many time understanding what should be done. Thus, users usually lose efficiency and effectiveness, increasing the losses of their corporations. Our framework aims to improve the usability of user when writing documentation. For instance, it is possible to send spreadsheets to other users without the need to explain it

because it is already documented. Thus, users will have less difficulties working on spreadsheets that were not created by them.

In [20] the author discusses for two kinds of documentation: development documentation and user documentation. The former is about the software itself, its internal form, and is created for developers, with technical knowledge about the software and its implementation details. The later is for the software users, with possibly no technical skills to understand documentation for developers. We have also separated both these kinds of documentation in SpreadsheetDoc. On one hand formulas should be documented technically, that is, with enough technical detail so they can be updated by other developers. On the other hand, input and output cells should be documented for end users so they can know where to input their values and read the results.

Abraham et al. developed a framework called UCheck [1]. This framework can compute the labels that affect each cell. Thus, when using the *headers* functionality the tool displays arrows directed from the header cells (labels) to the target cells, that is, the ones labelled by such text. The *units* functionality runs the unit checker and the system marks the cells with unit errors. For instance, a cell that adds cells with label apples and cells with label oranges is probably wrong. This work can also be seen as an automatic way of documenting spreadsheets, as it more explicitly shows information about the relationship of cells and their labels. It is interesting and relevant to our work because, by automatically inferring the headers, we could use this to show to users the correspondent labels of selected cells. Thus, we intend to integrate the inference system in our framework providing more information to users when they document spreadsheets, and specially when they read it. The more information we can provide to users, the more easily they will understand the spreadsheet they are working on.

7 Conclusions

Spreadsheets are the most used programming environments in the world. However, they lack many of the features modern programming environments offer. In particular, there is no structured way of documenting spreadsheet programs. Indeed, there is strong evidence that users wast too much time trying to understand spreadsheets, specially when they are using one that was not created by them. They search within spreadsheets trying to find some kind of documentation, ask for help to colleagues, or end up by quitting.

To alleviate this scenario, in this paper, we have presented a tool, SpreadsheetDoc, to document spreadsheets. It is built as an Excel add-in so users can easily install and use it. For each part of the spreadsheet SpreadsheetDoc offers the correct wizard with the necessary fields to be filled in by the user. The user can then read the documentation within the spreadsheet, but also in a web page generated from such documentation. It is also possible to export the documentation via an XML file so it can be reused by other tools. In fact, it is also possible to import documentation from an XML file.

7.1 Future Work

The work we present in this paper is the very first step towards easing the creating of documentation for spreadsheets. There are however many important directions for future work.

The most important one is the validation of our approach. Although we believe that it can help users to be more productive, we do not yet have the empirical evidence of such fact. In fact, there are at least two kinds of evaluations we intend to do. First, we intend to evaluate the usability of our tool, trying to learn if end users understand the purpose of each of its parts, and can actually use it in the correct way. Second, we will investigate its impact in users productivity, that is, given the same spreadsheet, one documented with our tool, and another not documented, or even documented with the current ad-doc techniques, investigate if users more easily and faster can understand the spreadsheet documented with SpreadsheetDoc. These evaluations will be performed with empirical studies, which we will design and run based on our previous experience [2, 3, 6].

Another quite interesting and promising direction is the automation or inference of documentation. As we mentioned, we intend to integrate the inference of headers and units [1], which will automatically give some documentation for users. For instance, for our running example, we would like to automatically document the input of cell F5 as being `Bid`, and not as being cell F4 (as `Bid` it the label of cell E4), since the label is much more informative than the cell reference. Moreover, we would also like to integrate some heuristics to automatically describe existing formulas, in a similar way as described in [17] for mining business rules from spreadsheets. For instance, for our running example, the system could automatically infer the following description for cell I10: *Cell I10 calculates Win? No, that is, the maximum between Profit new ship and Profit tug/barge.* This can easily be inferred from the formula, its inputs, and corresponding labels. Again, we have some experience in automatically inferring information from spreadsheet [4, 5], which will help us succeed in this work.

In this first approach we did not work on documenting the visual basic for applications (VBA) scripts that are part of some spreadsheets. Although this may seem important, in the Enron's corpus only 47 spreadsheets (out of more than 15.000) used VBA scripts [9]. Also in the EUSES corpus only 126 (out of 4.498) used VBA [8]. Nevertheless, we will also address this in future work. Since in this case we are probably addressing more advanced users, we intend to follow an approach similar to JavaDoc, where the programmer annotates the different parts of the source code, and from which it is possible to generate a comprehensive web page. This will extend the web page generated by our tool.

References

1. R. Abraham and M. Erwig. Header and unit inference for spreadsheets through spatial analyses. In *Proc. of the 2004 IEEE Symposium on Visual Languages and Human Centric Computing*, pages 165–172, Washington, DC, USA, 2004. IEEE.

2. L. Beckwith, J. Cunha, J. P. Fernandes, and J. Saraiva. An empirical study on end-users productivity using model-based spreadsheets. In S. Thorne and G. Croll, editors, *Proc. of the EuSpRIG Annual Conference*, pages 87–100, 2011.
3. L. Beckwith, J. Cunha, J. P. Fernandes, and J. Saraiva. End-users productivity in model-based spreadsheets: An empirical study. In M. Costabile, Y. Dittrich, G. Fischer, and A. Piccinno, editors, *Proceedings of the Third International Symposium on End-User Development*, pages 282–288, Heidelberg, June 2011. Springer.
4. J. Cunha, M. Erwig, J. Mendes, and J. Saraiva. Model inference for spreadsheets. *Journal of Automated Software Engineering (ASE)*, pages 1–32, 2014. in press.
5. J. Cunha, M. Erwig, and J. Saraiva. Automatically inferring ClassSheet models from spreadsheets. In *Proc. of the 2010 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 93–100, Washington, DC, USA, 2010. IEEE.
6. J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva. Embedding, evolution, and validation of spreadsheet models in spreadsheet systems. *IEEE Transactions on Software Engineering*, 43(3):241–263, 2014.
7. K. A. de Graaf, P. Liang, A. Tang, and H. van Vliet. Supporting architecture documentation: A comparison of two ontologies for knowledge retrieval. In *Proc. Int. Conf. on Evaluation and Assessment in Soft. Eng.*, pages 3:1–3:10. ACM, 2015.
8. M. Fisher and G. Rothermel. The EUSES spreadsheet corpus: A shared resource for supporting experimentation with spreadsheet dependability mechanisms. In *Proc. of the First Workshop on End-user Soft. Eng.*, pages 1–5. ACM, 2005.
9. F. Hermans and E. Murphy-Hill. Enron's spreadsheets and related emails: A dataset and analysis. In *Proceedings of the International Conference on Software Engineering (ICSE'15)*, Firenze, Italy, May, 2015. (to appear).
10. F. Hermans, M. Pinzger, and A. van Deursen. Breviz: Visualizing spreadsheets using dataflow diagrams. *CoRR*, abs/1111.6895, 2011.
11. F. Hermans, M. Pinzger, and A. van Deursen. Supporting professional spreadsheet users by generating leveled dataflow diagrams. In *Proc. of the 33rd International Conference on Software Engineering*, pages 451–460. ACM, 2011.
12. B. Klimt and Y. Yang. Introducing the Enron corpus. In *CEAS 2004 - First Conference on Email and Anti-Spam*, 2004.
13. J. Lawrance, R. Abraham, M. M. Burnett, and M. Erwig. Sharing reasoning about faults in spreadsheets: An empirical study. In *2006 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 35–42. IEEE, 2006.
14. S. G. Powell and K. R. Baker. *The Art of Modeling with Spreadsheets*. John Wiley & Sons, Inc., New York, NY, USA, 2003.
15. K. Rajalingham, D. R. Chadwick, and B. Knight. Classification of spreadsheet errors. *CoRR*, abs/0805.4224, 2008.
16. J. G. Raymond. Audience identification for end user documentation. In *Proceedings of the June 7-10, 1982, National Computer Conference*, AFIPS '82, pages 281–285, New York, NY, USA, 1982. ACM.
17. S. Roy. Business rule mining from spreadsheets. In F. Hermans, R. F. Paige, and P. Sestof, editors, *Proceedings of the Second Workshop on Software Engineering Methods in Spreadsheets*, volume 1355 of *SEMS '15*, pages 5–6. CEUR, 2015.
18. J. E. Scott. Technology acceptance and erp documentation usability. *Commun. ACM*, 51(11):121–124, Nov. 2008.
19. R. M. Snyder. A system for automating the update of spreadsheet documentation. *J. Comput. Sci. Coll.*, 23(2):163–169, Dec. 2007.
20. B. van Loggem. Software documentation: A standard for the 21st century. In *Proceedings of the International Conference on Information Systems and Design of Communication*, ISDOC '14, pages 149–154, New York, NY, USA, 2014. ACM.

Linguagem de manipulação de dados para NoSQL

Bruno Grácio[1], João Costa Seco[2] e Hugo Lourenço[3]

[1] Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
[2] NOVA LINCS - Universidade Nova de Lisboa
[3] OutSystems SA

Resumo O sucesso recente dos sistemas de base de dados *NoSQL* está muito ligado ao aparecimento de tecnologias de processamento de grandes quantidades de dados e de aplicações que usam um paradigma de computação em nuvem. A vantagem do uso destes sistemas reside na flexibilidade, escalabilidade horizontal e grande disponibilidade dos seus dados. Contudo, à medida que a popularidade destes sistemas cresce também a inexistência de uma linguagem standard, tal como o *SQL* para os sistemas relacionais, surge como um obstáculo na construção de interrogações. As interrogações são muitas vezes definidas utilizando linguagens imperativas, tornando o processo de desenvolvimento mais demorado e mais suscetível à introdução de erros de execução. Neste artigo, é proposta uma estratégia de compilação da linguagem de manipulação de dados da plataforma *OutSystems* para *NoSQL*, apresentando um modelo e uma infraestrutura de compilação que gera código *MongoDB*. A abordagem proposta destina-se a simplificar todo o processo de construção de interrogações *NoSQL*, tornando a construção de interrogações mais acessíveis a programadores inexperientes e seguras, pelo uso de um sistema de tipos que garante a consistência das mesmas.

Palavras-chave: NoSQL, Agregado, LDE, OutSystems, MongoDB

1 Introdução

O avanço tecnológico das aplicações web, dos dispositivos móveis e dos vários sensores que estes incluem resultaram num grande aumento da quantidade de dados que são guardados e processados, levando ao surgimento de conceitos como *Big Data* e computação em nuvem. Um exemplo de *Big Data* é o acelerador de partículas (*LHC*), que contém um fluxo de dados (das quatro experiências realizadas) que rondam os 700 *MB/s* o que equivale a cerca de 15 *PB* por ano [9].

Os sistemas de base de dados *NoSQL* ressurgiram para lidar com esta nova realidade, uma vez que estes estão melhor adaptados para cenários que envolvam grandes quantidades de dados, escalabilidade horizontal e para processamento eficiente. Estes sistemas representam um conjunto de sistemas de base dados com diferentes tipos e metodologias, podendo se classificar em quatro tipos: chave-valor, documentos, tabular e grafos.

Apesar das diferenças, os sistemas *NoSQL* têm em comum o facto de não seguirem o modelo relacional e de tentarem melhorar a escalabilidade e disponibilidade em compromisso de assegurar as tradicionais garantias *ACID* (Atomicidade, Consistência, Isolamento e Durabilidade). O modelo *ACID* funciona bem num sistema relacional, mas apresenta dificuldades em grandes sistemas distribuídos onde a disponibilidade e a eficiência são o aspeto principal [15]. Devido a tais características e aos sistemas *Big Data* e de Computação em nuvem [11], muitas empresas [8, 4] já estão a criar/migrar as suas aplicações para *NoSQL*, contudo este processo não é fácil.

O mercado *NoSQL* está extremamente fragmentado estando disponíveis muitas tecnologias e fornecedores de serviços distintos. Dentro da mesma tecnologia existem diferentes maneiras de estruturar os dados, diferentes linguagens de interrogação e diferentes mecanismos de pesquisa e manipulação dos dados, o que faz com que seja difícil a um programador a transição entre sistemas *NoSQL*. Em [14], os autores defendem que os programadores preferem despender o seu tempo a desenvolver aplicações, em vez de o gastar a aprender novas linguagens. Ou seja, os autores afirmam que as linguagens de interrogação *NoSQL* devem ser simples, reconhecíveis ao programador e acessíveis nas tecnologias com que os programadores já se encontram familiarizados.

A definição de uma linguagem de interrogação comum entre os sistemas *NoSQL* pode ser suficiente para convencer cada vez mais empresas, programadores e instituições a investir nestes sistemas. Neste artigo é apresentada uma estratégia de compilação da linguagem de manipulação de dados da plataforma *OutSystems*, linguagem dos Agregados [12], tendo como alvo bases de dados *NoSQL*, mais concretamente *MongoDB*. Esta proposta contribui para que os programadores *OutSystems* tenham uma experiência na definição de interrogações, independente da natureza da base de dados utilizada. A modularização promovida no mapeamento da linguagem *OutSystems* para *MongoDB* serve de base à utilização de outras tecnologias de base de dados. Este trabalho estende a linguagem modelo apresentada em [6] com a linguagem de manipulação de dados, dado que esta descreve a estrutura principal de uma aplicação *OutSystems*.

A *OutSystems* é uma empresa que tem como produto uma plataforma para o desenvolvimento de aplicações *web* empresariais utilizando uma *LDE* (Linguagem de Domínio Específico) integrada, que permite abstrair muitos dos detalhes de desenvolvimento e publicação de aplicações.

2 Agregados OutSystems

A linguagem de manipulação de dados na plataforma *OutSystems*, é uma *LDE* [10, 12] visual especialmente desenhada para a obtenção de dados num determinado sistema de base de dados. A construção de interrogações nesta linguagem é realizada através de um modo visual, onde o programador realiza uma série de operações sobre os dados até definir a interrogação pretendida. É importante referir que a linguagem que será apresentada é apenas a representação textual da *LDE* dos Agregados, sendo que o seu propósito não é ser utilizada pelo

$$
\begin{array}{rcl}
Agg & ::= & \textsf{aggregate } a \textsf{ \{ } Op \textsf{ \}} \\
\\
Op & ::= & \textsf{addsource } c \\
 & | & \textsf{project } \overline{type} \textsf{ \{ } Op \textsf{ \}} \\
 & | & \textsf{graft \{ } \overline{Src} \quad \overline{GraftCond} \textsf{ \}} \\
 & | & \textsf{group } details? \textsf{ \{ } Op \; \overline{Attr} \textsf{ \}} \\
 & | & \textsf{combine \{ } \overline{Src} \quad \overline{JoinCond} \quad \overline{Attr} \quad \overline{Filter} \quad \overline{Sort} \textsf{ \}} \\
\\
Src & ::= & \textsf{source } s? \textsf{ \{ } Op \textsf{ \}} \\
\\
JoinCond & ::= & s_1 \textsf{ (inner | outter | left | right) join } s_2 \textsf{ on } exp \\
GraftCond & ::= & \textsf{include } s_2 \textsf{ in } s_1 \textsf{ on } exp \\
\\
Filter & ::= & \textsf{filter } exp \\
Attr & ::= & \textsf{(calc | group | aggr) attr } a = exp \\
Sort & ::= & \textsf{sort } exp \textsf{ (ascending | descending)}
\end{array}
$$

Figura 1: Representação textual do modelo de dados dos Agregados.

programador. A representação abstrata proposta é independente da natureza das fontes de dados relacionais, não relacionais ou apenas dados em memória.

A Figura 1 representa a sintaxe da linguagem dos Agregados, onde um Agregado é representado por uma árvore de operações (*Op*) composta por nós *Project*, *CombineSource*, *GraftSource*, *GroupBy* e *AddSource*. Considerando-se que existe uma base de dados que contém dados relativos a alunos e cursos, como seria realizada a seguinte interrogação:

Quais os cursos cuja média de idades dos alunos é superior a 25 anos?

A resposta à questão anterior encontra-se representada na Listagem 2, onde se pode visualizar um exemplo de interrogação. A raiz de uma interrogação é sempre definida por um nó **Aggreagte**, que contém a árvore de operações sobre os dados e um identificador que é usado para o caracterizar nas *Widgets* do ecrã de uma aplicação *OutSystems*.

A operação **CombineSource** é utilizada para realizar filtros, definir atributos calculados, realizar ordenações e juntar informação de fontes de dados distintas. Para definir os campos pelos quais se pretende agrupar a informação ou aplicar funções de agregação, utiliza-se a operação de **GroupBy**. Atualmente as funções de agregação suportadas no protótipo são: *Max*, *Min*, *Count*, *Avg* e *Sum*. Por último, existe um operação de **AddSource** que contém uma referência para a coleção específica na base de dados e para a informação relativa à estrutura e aos tipos dos seus dados.

Listagem 2: Quais são os nomes dos cursos em que a média de idades dos alunos é superior a 25 anos?

```
aggregate GetStudentsWithOrWithoutCourses {
    combine {
        source {
            group {
                combine {
                    source Student { addsource Student }
                    source Course { addsource Course }

                    Student inner join Course on
                    ( Student.course_code ==
                      Course.course_code )

                    calc attr age =
                    ( Year(CurrDate()) -
                      Year(Student.birthday) )
                }

                group attr name = Course.name
                aggr attr ageAvg = Avg(age)
            }
        }

        filter (ageAvg > 25)
    }
}
```

A sintaxe atualmente suportada neste protótipo possui operações de *Project*, *GraftSource* e de *GroupBy* com detalhes. A operação **GraftSource** é uma nova operação introduzida no contexto de base de dados *NoSQL*, em que esta funciona como uma junção tradicional *SQL* mas onde não se preserva repetições. O seu resultado é uma lista em que cada elemento tem os dados que respeitam a condição de junção, sem repetições. As operações de *Project* e de *GroupBy* com detalhes, são operações que atualmente são definidas pelo programador à mão mas que na versão futura do protótipo serão introduzidas automaticamente pelo o otimizador. A operação **Project** pretende definir quais os campos que são usados no resultado de uma sub-árvore de operações. A operação de **GroupBy com detalhes**, referida acima com a palavra chave *group*, indica se se deve preservar os detalhes do *GroupBy* e qual a etiqueta associada a esses dados. Preservar os detalhes significa manter os campos que geraram a condição de *GroupBy*, ao contrário dos sistemas *SQL* que os perdem.

Na Figura 1 não se encontram definidas as regras sintáticas *type* e *exp*, contudo a lista de tipos que estes suportam são: *Integer*, *Text*, *Date*, *Boolean*, *Structure*, *List* e *Any*. Ao contrário da linguagem *JavaScript* a linguagem apresentada

é tipificada estaticamente. Isto ajuda o sistema de tipos a garantir que as interrogações definidas se encontram bem formadas, que o tipo de uma operação ou de uma expressão coincide com o tipo esperado no seu contexto de uso e que o mesmo não provoca qualquer comportamento inesperado ao protótipo.

3 MongoDB

A utilização do sistema *MongoDB* como base de dados *NoSQL*, deve-se ao facto deste ter ganho uma grande popularidade nos últimos anos. Este sistema usa um motor de base de dados que fornece operações como o *Aggregate* e *MapReduce*, que são utilizadas para mapear o comportamento da linguagem proposta.

3.1 Operação Mongo *Aggregate*

Aggregate é uma operação que funciona com base num modelo de agregação de dados processados em *pipeline* e a sua estrutura está representada na Listagem 3. A *pipeline* do Agregado, consiste nas seguintes etapas:

- **Project:** Transforma os documentos, adicionando novos campos ou removendo campos já existentes. Função útil quando se pretender introduzir *Sources* e atributos calculados.
- **Match:** Permite filtrar os documentos que passam à próxima etapa da *pipeline*. Para cada documento, aplica-se uma condição de filtro que indica se o mesmo passa para a próxima etapa da *pipeline*.
- **Unwind:** Decompõe um campo do tipo lista do documento fornecido por *input* e por cada elemento dessa lista, fornece um novo documento com esse elemento e os restantes campos. Por exemplo, se um documento contiver um campo com a idade (ex: 20 anos) e outro campo com pessoas (ex: [Pedro,Rita]), se aplicarmos esta função ficamos com pares (Pedro,20) e (Rita,20).
- **Group:** Agrupa cada documento pelo identificador e aplica funções de agregação aos campos pretendidos. Esta função fornece os identificadores agrupados e as funções de agregação aplicadas aos campos.
- **Sort:** Reordena os documentos fornecidos por um campo específico.
- **Out:** Guarda os resultados da *pipeline* numa coleção temporária. Esta operação deve ser utilizada no final da *pipeline*.

Listagem 3: Estrutura da operação de *Mongo Aggregate*.

```
1 db.COLLECTION.aggregate([ STAGE 1, STAGE 2, ..., STAGE N ])
```

Cada etapa transforma o documento à medida que este passa através da *pipeline*, sem ser necessário produzir um documento de *output* por cada entrada. Algumas etapas podem produzir novos documentos ou filtrar os mesmos. As etapas podem aparecer repetidas na *pipeline* e as suas expressões apenas podem referenciar informação que esteja contida nos documentos fornecidos como *input*.

Uma limitação deste *Aggregate* é o facto de só funcionar sobre uma única coleção, passando todo o conteúdo do mesmo para a *pipeline*. Esta operação também permite a otimização das operações definidas dentro da sua *pipeline* com o recurso a índices que devem estar previamente criados. Um exemplo de otimização são as operações de *Match* e *Sort* que podem tirar vantagem de índices. Uma filtragem precoce pode também melhorar a eficiência das etapas seguintes, pois reduz o número de documentos a passar através da *pipeline*.

3.2 Operação Mongo *MapReduce*

O *MapReduce* é um modelo de programação que permite condensar grandes volumes de dados. Esta operação vai ser utilizada no protótipo para ultrapassar a dificuldade da junção de informação em fontes de dados *NoSQL*. Apesar deste caso ser pouco comum em sistemas *NoSQL*, no contexto deste artigo deve-se considerar esta possibilidade. Em *NoSQL* não existe um mapeamento direto desta operação, ao contrário dos sistemas *SQL*, pelo que para isso será necessário definir as seguintes funções:

- **Map:** Função de *Javascript* que mapeia um valor com uma chave e emite-os para as funções de *Reduce*.
- **Reduce:** Função de *Javascipt* que reduz os valores emitidos pela função *Map*, num único resultado.
- **Out:** Especifica a coleção temporária onde vão ser guardados os resultados após a aplicação das funções de *Map* e *Reduce*.
- **Finalize:** Função *Javascript* que é aplicada a seguir à função de *Reduce*. Esta função, modifica o *output* dos *Reduces* que vão sendo guardados numa coleção temporária.

Listagem 4: Estrutura da operação de *Mongo MapReduce*.

```
1 db.COLLECTION.mapReduce( MAP, REDUCE, {
2     out: TEMP_COLLECTION,
3     finalize: FINALIZE
4 })
```

A estrutura desta operação está representada na Listagem 4, sendo importante referir que todo o código *JavaScript* definido nas funções de *Map*, *Reduce* e *Finalize* é realizado do lado do servidor, tirando essa complexidade do cliente. Atualmente o protótipo apenas está preparado para realizar operações do lado do servidor, mas num futuro próximo pretende-se que seja possível a realização de novas operações em memória [13].

De uma forma geral, a operação de *MapReduce* aplica a função de **Map** a cada documento onde são emitidos pares chave-valor. Para as chaves com múltiplos valores, é aplicada uma função de **Reduce** que processa a informação e condensa a mesma guardando-a para uma coleção intermédia. Opcionalmente é chamada uma função de **Finalize** após a execução de todos os *Reduces*, ou após a execução da função *Map* para chaves que tenham um único valor.

3.3 Exemplo de uma interrogação MongoDB

O mapeamento da interrogação *OutSystems* apresentada previamente na Listagem 2 para uma interrogação *MongoDB* é formada pela seguinte sequência de interrogações representadas nas Listagens 5, 6 e 7.

Listagem 5: Operação de *MapReduce* sobre a coleção de alunos.

```
db.students.mapReduce(
    function() {
        var Student = {city:this.city, ...};
        emit({f0:Student.course_code},{city:this.city, ...});
    },
    function(key,values) {
        var result = {Student: []};

        values.forEach(function(value) {
            if(value.city != null || ...)
                result.Student.push(value);

            if(value.Student != null)
                result.Student =
                    result.Student.concat(value.Student);
        })

        return result;
    },
    {out:"temp_student_course",finalize:function(key,value) {
        if(value.Student == null) {
            var result = {Student: []};
            result.Student.push(value);
            return result;
        }

        return value;
    }})
```

Listagem 6: Operação de *MapReduce* sobre a coleção de cursos.

```
db.courses.mapReduce(
    function() {
        var Course = {cc:this.cc, ...};
        emit({f0:Course.cc},{cc:this.cc, ...});
    },
    function(key,values) {
        var result = {Course: [], Student: []};

        values.forEach(function(value) {
            if(value.cc != null || ...)
                result.Course.push(value);
```

```
12
13              if(value.Course != null)
14                  result.Course =
15                      result.Course.concat(value.Course);
16
17              if(value.Student != null)
18                  result.Student =
19                      result.Student.concat(value.Student);
20          })
21
22          return result;
23      },
24      {out:{reduce:"temp_student_course"}})
```

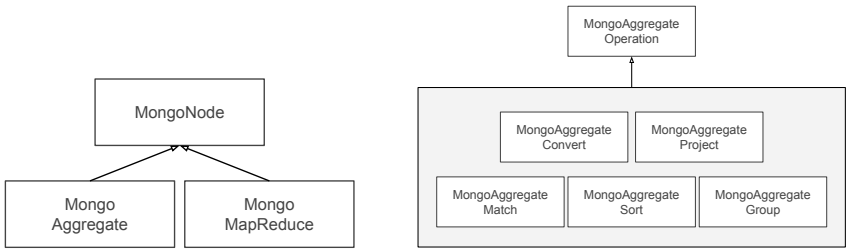
Listagem 7: Operação de *Aggregate* sobre a coleção temporária produzida pelas funções *MapReduce* representadas na Listagens 14 e 15.

```
1  db.temp_student_course.aggregate([
2      {$project:{
3          Student:"$value.Student",Course:"$value.Course"
4      }},
5      {$unwind:"$Student"},
6      {$unwind:"$Course"},
7      {$project:{
8          Student:{city:"$Student.city", ...},
9          Course:{cc:"$Course.cc", ...}
10     }},
11     {$group:{
12         _id:{nameA:"$Course.course_name"},
13         ageAvg:{$avg:"$age"}, ...
14     }},
15     {$project:{
16         nameA:"$nameA",ageAvg:"$ageAvg",
17         f0:{$gt:["$ageAvg",25]}
18     }},
19     {$match:{f0:true}}
20 ])
```

4 Mongo Adapter

Para atingir uma melhor modularidade, a plataforma *OutSystems* usa uma arquitetura com adaptadores para abstrair todo o processo de geração de código para um determinado sistema de base de dados, facilitando assim o processo de suportar outros sistemas. Com esta abstração, caso se pretenda suportar outros sistemas de base de dados basta criar o respetivo adaptador da linguagem específica, respeitando sempre uma estrutura predefinida. Esta estrutura é

composta por um método que indica qual o *Driver* necessário para executar interrogações de uma determinada linguagem, um método **Generate Aggregate** que processa o Agregado e devolve a interrogação correspondente e um método **Get Temporary Collections** que devolve a lista de coleções temporárias, uma vez que o processo de junção de coleções em ambiente não relacional pode conter representações intermédias.



(a) Possíveis nós da árvore *MongoNode*

(b) Possíveis nós da árvore *MongoAggregateOperation*

Figura 8: Árvores de operações associadas ao objeto *MongoAdapter*

O processo de mapeamento da representação textual do Agregado *OutSystems* para *MongoDB* é realizada com a ajuda de uma árvore auxiliar, que representa as interrogações *MongoDB* necessárias para obter os dados pretendidos. Esta árvore auxiliar é do tipo **MongoNode** (Figura 8) em que cada nó num dado momento pode ser um *Mongo Aggregate* ou um *Mongo MapReduce*. Existe a necessidade de utilizar uma representação em árvore devido a existirem operações em que é necessário uma representação intermédia dos dados (operações de junção ou preservar detalhes em operações *GroupBy*), onde existe um nó *Mongo MapReduce* aplicado a dois filhos do tipo *Mongo Aggregate*.

No nó **Mongo Aggregate** foi criada uma *pipeline* de operações, sendo que esta pode ser vista como uma árvore degenerada de etapas, criando assim uma representação extensível para futuras operações e simplificada para o processo de otimização. Todas as etapas desta *pipeline* são equivalentes às etapas da *pipeline* do sistema *MongoDB* apresentada na Secção 3.1, exceto a etapa *Mongo Aggregate Convert*. Esta, é responsável pela conversão da representação intermédia do *Mongo MapReduce* para a representação que o programador pretende, ou seja, se for realizada uma junção de duas coleções o *Mongo Aggregate Convert* faz *Unwind* (Secção 3.1) e um *Project* dos campos das duas coleções de forma a obter os dados da mesma forma que o *SQL*.

5 Implementação

Nesta secção é explicado todo o processo de mapeamento da linguagem de Agregados para interrogações *NoSQL*. Os exemplos apresentados nesta secção são tipicamente relacionais, onde será utilizada a base dados de uma Faculdade com uma tabela de alunos (Tabela 10) e uma tabela de cursos (Tabela 9). A Listagem 2 contém a representação textual de um Agregado, onde se pretende obter os nomes dos cursos em que os seus alunos têm uma média de idades superior a 25 anos. A árvore sintática gerada para a respetiva interrogação está representada na Figura 11a, onde se consegue verificar a hierarquia de operações realizadas.

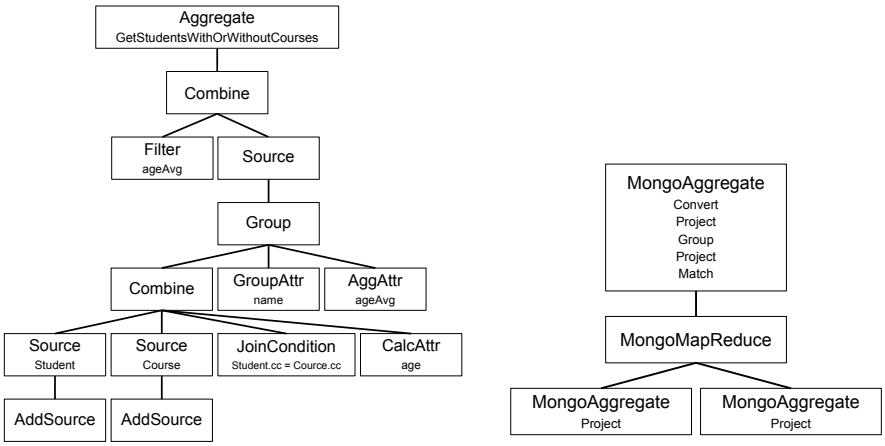
Tabela 9: Conteúdo da tabela/coleção cursos.
CC = Course Code

CC	**Course Name**
1	Engenharia Informática
2	Engenharia Eletrotécnica
3	Engenharia do Ambiente
4	Matemática
5	Arquitetura

Tabela 10: Conteúdo da tabela/coleção alunos.
CC = Course Code

Number	**Name**	**City**	**Birthday**	**Gender**	**CC**
1	Joaquim Pires Lopes	Lisboa	1994-12-31	M	2
2	Ana Maria Fonseca	Setubal	1990-02-28	F	1
3	Paula Antunes	Lisboa	1984-07-12	F	2
4	Joana Ramalho Silva	Seixal	1994-09-22	F	3
5	Rui Manuel Silva	Porto	1984-08-14	M	1
6	Joao Paulo Santos	Lisboa	1991-11-15	M	1
7	Cristina Fernandes Lopes	Lisboa	1996-01-06	F	1
8	Miguel Pinto Leite	Porto	1984-01-06	M	3
9	Francisco Costa Rosa		1992-02-15	M	4
10	Elsa Fialho Pinto		1989-10-28	F	1
11	Maria Pinto Ribeiro		1993-10-19	F	1

No processo de compilação de um Agregado, começa-se por enviar a árvore sintática representada na Figura 11a para o *Mongo Adapter*. Este navega por cada nó da árvore de operações e produz código *MongoDB* correspondente até gerar a árvore auxiliar representada na Figura 11b. As operações *AddSource* referenciam fontes de dados distintas, e tal como foi explicado na Secção 3 uma das



(a) Árvore sintática gerada pela linguagem dos Agregados.

(b) Árvore gerada pelo *Mongo Adapter*.

Figura 11: Árvores geradas relativamente à Listagem 2

limitações deste sistema é o facto de as operações *MongoDB* só poderem ser aplicadas a uma coleção individualmente, assim o mapeamento direto das operações *AddSource* passa pela criação de operações *Mongo Aggregate* individuais em que a etapa da *pipeline* corresponde a uma projeção de todos os campos da coleção.

De seguida existe uma *Join Condition* entre duas *Sources* individuais, sendo necessário criar um nó de *Mongo MapReduce* que contém definido internamente duas funções de *MapReduce* isoladas, que são executadas sequencialmente em cada coleção guardando os resultados numa coleção intermédia. Ao processar a condição de junção, verificam-se quais as *Sources* presentes e criam-se as respetivas funções *MapReduce*. Este algoritmo é baseado em [2, 5].

Na Figura 14 encontra-se a primeira função de *MapReduce* que é necessária executar sobre a coleção alunos de forma a conseguir mapear a junção em *MongoDB*. Apenas é necessário definir as funções de *Map*, *Reduce* e *Finalize* uma vez que o *MapReduce* encarrega-se de abstrair as restantes funções.

A função de *Map* deve respeitar a estrutura da Listagem 12. Neste caso particular a *Source Student* não contém operações complexas, bastando emitir o par código curso e campos da coleção.

A função de *Reduce* (Listagem 13) deve processar uma lista de valores com uma chave em comum e retornar apenas um valor. Neste caso particular, a função deve conseguir devolver uma lista de alunos na coleção temporária. Para isso, cria-se um objeto com um campo *Student* que irá ser uma lista de alunos e de seguida itera-se sobre os valores de entrada guardando-os no respetivo objeto. Deve-se considerar o facto de o *Reduce* poder ser chamado mais do que uma vez e algumas dessas vezes com resultados intermédios.

Listagem 12: Esqueleto de uma função de *Map* da função *MapReduce*

```
function()
{
    //Reconstruir Sources
    var Source1 = {name:this.name, ...};
    var Source2 = {Source1:Source1,name2:Source1.name,...};

    //Inserir filtros aqui
    if (Source2.name2 != "")
    {
        //Emitir dos pares (key,value)
        emit( {f0:Source2.id},
              {Source1:Source1,name2:Source1.name,...} );
    }
}
```

A função *Finalize* é semelhante à função de *Reduce*, uma vez que esta é invocada no final para tratar dos dados antes que os mesmos sejam armazenados numa coleção temporária. Existe um caso particular que faz com que esta função realize algum trabalho extra quando a função *Map* apenas emite um valor, uma vez que nestas condições a função *Reduce* não é invocada.

Na Figura 15 encontra-se definida a segunda função *MapReduce* para o processo de mapeamento de uma junção em *MongoDB*. Tal como a primeira função de *MapReduce*, também aqui se devem definir as funções de *Map*, *Reduce* e *Finalize*. A definição destas funções é semelhante ao exemplo anterior, frisando apenas que neste caso não é necessário implementar a função de *Finalize*, uma vez que o tratamento dos dados será todo realizado na função de *Reduce*.

Após a execução da segunda função de *MapReduce*, irá haver um problema de consistência dos dados, pois a coleção temporária já contém os resultados da execução da primeira função *MapReduce*. De forma a resolver esta questão, aplica-se novamente a função de *Reduce* para realizar a união dos resultados na coleção temporária como mostra a Figura 16.

A execução de um nó *Mongo MapReduce* (Figuras 14, 15, 16) cria uma representação intermédia que ainda não está com a representação desejada. Para resolver esta questão, cria-se um nó *Mongo Aggregate* com uma operação de *Convert*. Esta operação será responsável por converter a representação intermédia produzida pela função de *Mongo MapReduce* para uma representação igual aos sistemas relacionais, preservando as repetições.

Voltando à árvore sintática da Figura 11a, verifica-se que após a compilação da condição de junção existe um atributo calculado. Para a compilação desse atributo, será introduzido na *pipeline* um *project*. De seguida encontra-se um nó *GroupBy*, que para ser compilado basta criar uma operação de *group* na *pipeline* do *Mongo Aggregate* e introduzir aqui os atributos pela qual se pretende agrupar e os atributos que se pretendem agregar.

Listagem 13: Esqueleto de uma função de *Reduce* da função *MapReduce*

```
function(key,values)
{
    var result = {Source3:[], ..., Attr1, ...};

    values.forEach(function(value)
    {
        //Testa se os valores são os da função Map
        if(value.Source1 != null || ...)
        {
            result.Source3.push(value);
            //Inserir atributos agregados aqui
        }

        if(value.Source3 != null)
        {
            result.Source3 = result.Source3.concat(...);
            //Combinar atributos agregados aqui
        }

        //Testa se a Source4 já existe na coleção temporária
        if(value.Source4 != null)
        {
            result.Source4 = result.Source4.concat(...);
        }
    })

    return result;
}
```

A compilação do filtro definido na operação *CombineSource*, introduz as próximas duas operações de *project* e *match* na *pipeline*. Outra limitação do *MongoDB* prende-se com o facto de os filtros só poderem ser aplicados a um campo da coleção. Ou seja, se a condição de filtro for complexa quer do lado direito como do lado esquerdo do operador, não existe maneira de mapear esse comportamento diretamente com a operação de *match*. Para isso, introduz-se a condição de filtro como um atributo calculado, sendo possível no *match* verificar se esse atributo calculado é verdadeiro ou falso.

Finalmente, depois de produzida a árvore auxiliar, basta processar a mesma de forma a obter apenas as interrogações necessárias para obter os dados.

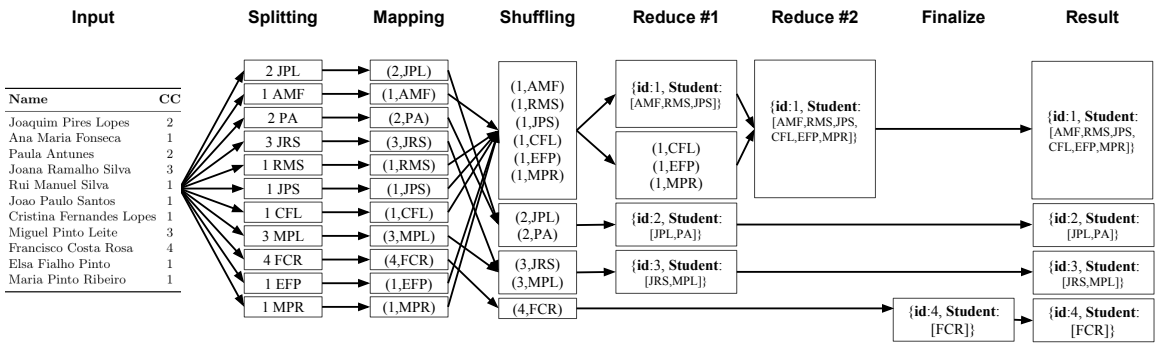


Figura 14: Fase 1 da função de *MapReduce*

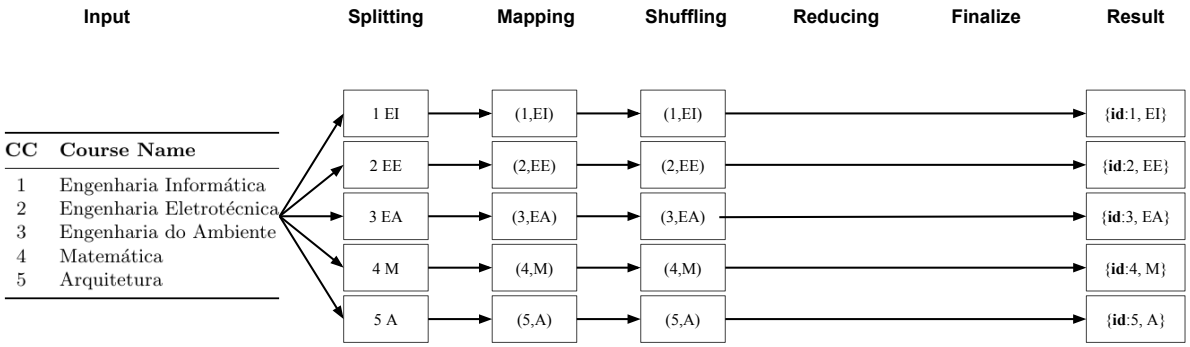


Figura 15: Fase 2 da função de *MapReduce*

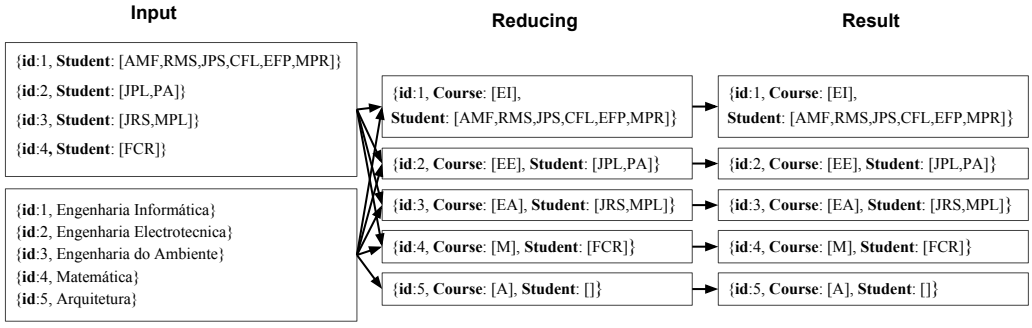


Figura 16: Fase 3 da função de *MapReduce*

6 Trabalho Relacionado

Existem operações que são muitas vezes impossíveis de se realizar em contextos de *Big Data* devido ao elevado consumo de memória e tempo que estas requerem. Em [2], é apresentado um exemplo de uma base de dados *NoSQL* que tira partido da função de *MapReduce* para fornecer uma forma eficaz de agregar uma grande quantidade de dados, sendo também apresentado neste artigo diversas formas de modelar eficientemente estas funções.

Em [1], é apresentado um cálculo para o processamento de dados semi-estruturados que suporta as diferenças existentes entre as linguagens *NoSQL*. Este cálculo permite aos programadores definir os seus próprios operadores sobre esses sistemas, com recurso a um algoritmo de inferência de tipos baseado num verificador semântico, o que resulta em informação flexível o suficiente para lidar com dados estruturados e semi-estruturados.

O *DataStax* [3] é uma tecnologia que integra um sistema de armazenamento de dados (*Hive*) com um sistema de base de dados *NoSQL* (*Cassandra*). O *Hive* disponibiliza uma linguagem muito parecida ao *SQL* (*HiveQL*) que abstrai todo o código de *Map Reduces*, disponibilizando operações de junção que o *Cassandra* como base de dados *NoSQL* não suporta.

Em [7], é proposta a criação de um sistema eficiente para a criação de interrogações *SQL* para sistemas de base de dados relacionais e para serviços *web*, permitindo a junção dos dados destas duas fontes de dados. Um dos inconvenientes encontrados pelo o autor foi o facto de não existir uma linguagem comum entre as base de dados relacionais e a *API* que os serviços *web* disponibilizam.

7 Conclusões e Trabalho Futuro

O trabalho apresentado neste artigo, mostra que é possível utilizar a *LDE* visual de manipulação de dados na plataforma *OutSystems* para lidar com sistemas de base de dados *NoSQL*. Contudo, nem todas as operações têm um mapeamento direto, sendo que atualmente no protótipo ainda existem algumas restrições na forma como o programador utiliza os Agregados *OutSystems* com dados *NoSQL*.

Uma das questões que ainda se pretende resolver é a otimização dos dados transferidos entre cliente-servidor, bem como a transferência de dados entre as várias etapas da *pipeline* de um Agregado em *MongoDB*. Atualmente quando é realizada uma interrogação sobre a linguagem de manipulação de dados na plataforma *OutSystems*, todos os campos do Agregado são processados e enviados para o cliente, no entanto todo este processo de geração da interrogação deve ser capaz de perceber que dados é que o programador pretende consumir de forma a restringir a quantidade que o mesmo recebe, otimizando a abertura da página e podendo assim disponibilizar aplicações rápidas e fluidas.

A utilização de informação relativa a índices na fase de compilação de interrogações é também um aspeto que necessita de algum trabalho. Se uma aplicação tiver conhecimento dos índices existentes numa dada base de dados, pode utilizar essa informação para construir interrogações mais eficientes. Um caso de

uso claro onde esta otimização teria um grande impacto, seria no momento da compilação de uma condição de junção para *MongoDB*. Atualmente considera-se que é sempre necessário aplicar funções *MapReduce* sobre as coleções definidas na condição de junção, no entanto esta pode não ser a melhor opção, porque se a primeira função *MapReduce* gerar poucos resultados e a segunda coleção for muito grande, então é mais eficiente fazer pedidos diretos à base de dados do que aplicar uma função *MapReduce* sobre a segunda coleção.

Referências

1. Véronique Benzaken, Giuseppe Castagna, Kim Nguyen, and Jérôme Siméon. Static and dynamic semantics of nosql languages. In *Proceedings of Symposium on Principles of Programming Languages*, POPL '13. ACM, 2013.
2. Laurent Bonnet, Anne Laurent, Michel Sala, Benedicte Laurent, and Nicolas Sicard. Reduce, You Say: What NoSQL Can Do for Data Aggregation and BI in Large Repositories. In *Proceedings of the International Workshop on Database and Expert Systems Applications*, DEXA '11. IEEE Computer Society, 2011.
3. DataStax. Datastax Enterprise 3.1 Doc - Using Hive. Acedido: 22/12/2014.
4. DataStax, San Francisco. *Ebay Engages Customers With Personalized Recommendations*, 2013. Case Study.
5. Nathan Ehresman. Tebros Systems: Using MongoDB MapReduce to Join 2 Collections. http://tebros.com/2011/07/using-mongodb-mapreduce-to-join-2-collections/. Acedido: 17/11/2014.
6. Sara Alpoim Gonçalves, João Costa Seco, Hugo Lourenço, and Sérgio Silva. Otimização Automática de Aplicações Web Usando Templates Client-Side. *INForum 2014 Atas do 6º Simpósio*, 2014.
7. Nuno Grade, Lúcio Ferrão, and João Costa Seco. Optimizing data queries over heterogeneous sources. *INForum 2013*.
8. Tyler Harter, Dhruba Borthakur, Siying Dong, Amitanand Aiyer, Liyin Tang, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Analysis of HDFS Under HBase: A Facebook Messages Case Study. In *Proceedings of File and Storage Technologies*, FAST'14. USENIX Association, 2014.
9. C Lefevre. CERN brochure (English version). Jul 2010.
10. Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and How to Develop Domain-specific Languages. *ACM Computing Surveys*, 2005.
11. A. M Moniruzzaman and S. Akhter Hossain. NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. *ArXiv e-prints*, June 2013.
12. OutSystems. Fetching data from the database. http://www.outsystems.com/help/servicestudio/9.0/Using_Data/aggregates/Fetching_Data_from_the_Database.htm. Acedido: 13/07/2015.
13. João Costa Seco, Hugo Lourenço, and Paulo Ferreira. A Common Data Manipulation Language for Nested Data in Heterogeneous Environments. *The 15th Symposium on Data Base Programming Languages*, 2015.
14. Timothy Stephan. What NoSQL Needs Most Is SQL. http://data-informed.com/what-nosql-needs-most-is-sql/. Acedido: 1/06/2015.
15. Meenu Dave Vatika Sharma. SQL and NoSQL Databases. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2012.

Social Media Integration in Video Games: A Social Overlay for Desktop Games

Joana Osório and Filipe Pacheco,

Polytechnic Institute of Porto (ISEP-IPP), Computer Engineering Department, Porto, Portugal
{1100594, ffp}@isep.ipp.pt

Abstract. The ever increasing popularity of social media makes them a promising source for the personalization of gameplay experiences. Furthermore, involving social network friends in a game can greatly enrich the satisfaction of the player and also attract potential novel players to a game. This paper describes a social overlay designed for desktop games. It allows players to easily capture and share on multiple social networks screenshots, videos and even game-related stories. Unlike most social sharing systems our social overlay is designed to interact with the user in a non-intrusive way allowing him/her to be in complete control of what is shared. Our goal is to make players look and ask for social integration. The development of this social overlay will allow players to take full advantage of their social communities to improve their gaming experience.

Keywords: Social media integration, Desktop games, Social overlay, Social networks, Gameplay experiences

1 Introduction

Gamers often compete, collaborate and share stories of gaming accomplishments. Over the years this leads to game developers realizing that they have to support the growth of gaming as a social hobby. This fact can also contribute to the creation of collaborative environments to improve the game quality of every participant [1]. However, social media integration can quickly evolve into a spam-like promotion system [2] for the players who connect their games to social networks accounts, so developers need to carefully consider the benefits and disadvantages of social media as an opt-in experience.

To promote social media integration in desktop games we have developed and tested a social overlay in the context of a small multiplatform game project, called Adventure! The Paladin Order (APO) [3], however it intends to be applied to different desktop game environments. Our social overlay is designed for desktop players of all age groups, but with a focus in young people (typically aged from 18 to 25) that are regular users of social networks like Facebook and Twitter [4]. With our overlay, players will always be able to choose not to use the social features, but if they opt to use them, they will have benefits such as the capacity to capture and share their game

experiences in different social networks and in the future, the creation of collaborative environments with social network friends for the personalization of their game experiences.

This paper describes a user-friendly, non-intrusive and multiplatform social overlay for desktop games to allow players to enhance their gaming experiences using the social communities. It is structured in 7 sections. Following this general introduction section 2 reviews the state-of-the-art of the related literature and compares the existing systems with our social overlay prototype. Sections 3-6 are devoted to present our social overlay including its requirement specifications, system design, prototype implementation and prototype alpha evaluation. Finally, in section 7 we summarize the conclusions and main future work.

2 Background on Social Media Integration

Full social media integration in games is often seen in online games that can be played through social networks. Usually these games include multiplayer features or asynchronous gameplay mechanics. These games are frequently implemented in the web browser or in mobile device apps [5]. Social network games are among the most popular games in the world, with games such as FarmVille [6], Mafia Wars [7], The Sims Social [8] and Candy Crush [9].

Recently, the gamers have realized that they can expand the gamming community directly through YouTube, Twitter, Facebook and other popular social media sites. Gamers now often record and upload their gameplays to YouTube to receive comments and share gaming strategies and principles. These new habits created business opportunities that are staring to be explored. For example, Achievement Hunter, a Rooster Teeth spinoff, is exclusively dedicated to the upload gameplay videos to the Let's Play YouTube Channel [10]. Live-streaming, particularly Twitch.Tv, has also become popular lately [11], it fostered the appearance of "gamer celebrities" that congregate in real time a large numbers of fans to watch and comment the game live. Gamer Sacriel was one of the top leaders on Twitch.tv in 2013 and he has a regular audience of more than 2,000 simultaneous viewers on his game streams. This has allowed him to become a professional gamer making his income solely on revenues from YouTube and Twitch.tv [12].

To address this new reality, social media integration in video games evolved to a new dimension and it is increasingly available on game consoles, such as Microsoft Xbox [13] or Sony's PlayStation 4 [14] and also on desktop games [15, 16]. Typically the social features for these types of platforms are media sharing, live broadcasting and exchanging playing experiences.

2.1 Strategies of social media integration

Kietzmann et al. [17] presented a model that stratifies different types of social media accordingly to the focus placed on seven functional blocks: identity, conversations, sharing, presence, relationships, reputation and groups. Exploring this model can help developers and companies to better design their social network targeting strategies.

Based on this model Facebook is used to promote long-term relationships with the target audience and can be integrated in almost all contexts. In this platform there is a strong connection between identity and relationships. In the gaming context Facebook-users have specific profiles for self-promotion (real identity vs. virtual identity) and share motivations, causes, events and activities with friends. On other hand, Twitter is more focused in the conversation than in the identity of users. In this platform, users "tweet" short and real-time messages which are often scores, real-time status updates or game stories with no obligation to answer. Kietzmann model can equally be applied to YouTube, which is centred in the sharing of previously recorded videos. Initially, YouTube only allowed users to upload homemade videos, but now is also used to improve the marketing of companies and for sharing game videos. Unlike Facebook that uses "likes" for reputation, YouTube reputation is based on "view counts" and "ratings". In this platform the group relationships really matter, so this type of platform is also indicated for user experience sharing.

2.2 Similar Systems and Prototype Progress

Presently, there are several successful stories of the use of social integration tools for game promotion and enhancement. In the last decade these tools vary greatly on its characteristics, complexity and dedicated financial investment.

One of the first relevant examples of game-related social media integration was revealed on November 17th of 2009 upon the presentation of Xbox 360. It had native applications for Facebook, Twitter and Last.fm [13]. The Facebook application allowed gamers to update their status, comment, like and view friends' pictures. Additionally, “Xbox Live Friend Finder” allowed gamers to identify Facebook friends that used Xbox Live [18]. For unclarified reasons, on October 16th 2012 Facebook and Twitter applications were removed from Xbox Live limiting users to access Facebook and Twitter through the Xbox 360 web browser [19].

In August 2011, Overwolf Ltd. announced a free social overlay for Windows desktop games [15] that included a variety of social features inside the game environment such as the ability to browse contents, share pictures and statuses on Facebook and Twitter (including game-related stories and pictures), upload and watch YouTube videos and make Skype calls. It also allowed game recording and live gameplay streaming to Twitch. Currently, Overwolf provides official support for more than 1000 different Windows desktop games [20] and a store with a wide range of free social network applications. However, some of these applications for example the ones for conversation or ongoing game detection are still limited in functionality comparatively to other social overlays and there is a complete lack of versions for other platforms other than Windows. In our social overlay we also provide support for the common share and screen capture actions, however to bypass the Overwolf compatibility limitation, our overlay was developed using Unity3D [21] and offers support to Windows, Mac OSX and Linux. Later in this year, Echobit released Evolve [22] a gamming platform for Windows desktop games with a hotkey accessible social overlay. Beyond the basic social overlay functions (e.g. gameplay sharing and chatting) this platform developed a new feature called “Party” that can be initiated for multiple games at the same time. It allows players to search the game statuses of their

friends and create parties (teams) in the games that they are playing. Presently, Evolve officially supports more than 4300 games, four social networks: Twitter, Facebook, YouTube and Twitch and for four platform distribution services: Steam, Battle, Origin and PSN [16]. The major limitation of Evolve beside the absence of a multiplatform version, is the lack of APIs and SDKs to easily customize and integrate this tool in games. Similar to this overlay, our prototype includes a “share” hotkey and allows integration of Twitter, Facebook and YouTube, which according to Ann Hurk [23] are the most used social media platforms. Contrary to Evolve, we can export our social overlay tools as Unity3D assets [24] and integrate them in different Unity3D development environments for desktop games.

More recently, Sony took a decisive step forward in social media integration in their games by incorporating a "Share button" in the back of the PlayStation 4 (PS4) game controller. This button was revealed during the worldwide presentation of PS4 on February 20th of 2013 [14] and allows players to easily share their gaming experiences directly to the PlayStation network, Facebook, Twitter, Twitch and USTREAM. This "Share" button is the first existing social feature of its kind and is analogous to our “share” hotkey, however instead of only share gameplay experiences, we will also provide asynchronous game mechanics to integrate social networks friends in the creation of new game elements and in the obtainment of different game bonus.

3 Requirements Specifications

The main goal of our social overlay is to enhance the gameplay experience of the players using social media communities as resource. The involvement of friends and communities can promote sharing of game strategies and achievements among players and avoid eventual irritation or boredom states in the player that may drive him to stop playing [25].

Requirements were divided into two types: functional requirements and non-functional requirements. The first type of requirements presents what the prototype should do and the second type describes how the prototype should work.

The functional requirements where prioritized into three categories (essential, conditional and optional), according to their importance for the prototype. Table 1 presents the prioritization scale of functional requirements, as well their descriptions.

Table 1. Prioritization scale of functional requirements.

Scale level	Description
Essential	Critical requirements, without them the product is not acceptable.
Conditional	These requirements would improve the product quality.
Optional	The requirements that would be nice to have, but are not indispensable to the product.

As the social overlay is being integrated in Adventure! The Paladin Order (APO) for tests purposes, the agreement on essential requirements was made informally. In what regards to conditional and optional requirements the prioritizing was based on the opinions of APO's test team that were collected using a survey. APO's test team are an external group of 10 young players with ages between 19 and 25 that were recruited by our Clockwork Inc. team [3] for the preliminary testing phases. We considered "optional requirements" all the requirements with average between 1 and 2.5. The requirements with average between 2.5 and 5 were considered "conditional requirements". The opinions of APO's test team about use case-based requirements related with "Broadcast Gameplay" have an average result of 3.5 (Fig.1) in a scale of 1 to 5. Regarding to the "Craft" use case-based requirements, their opinions have an average result of 4.1 (Fig.1) in a scale of 1 to 5. Finally, the opinions about use case-based requirements associated with "Use passive party system" have an average result of 2.1 (Fig.1) in a scale of 1 to 5.

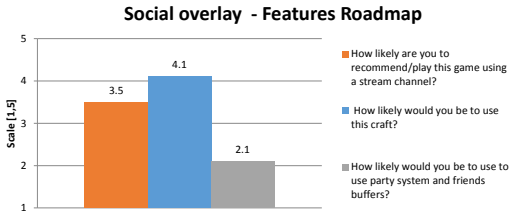


Fig. 1. Average of APO's test team opinions (scale of 1 to 5) about broadcast gameplay (orange), craft system (blue) and passive party system (gray).

Below we present our use case diagram, as well the prioritization scale of use case-based requirements.

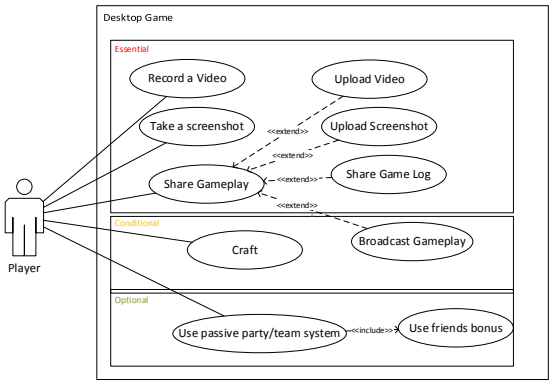


Fig. 2. Use case diagram of social overlay prototype with prioritization scale.

"Record video" and "Take a screenshots" use cases, classified as essential, allow players to record short videos (up to 15 minutes) and take screenshots during the

game. The resulting videos and screenshots can be uploaded to social networks using the “upload video menu” and “upload screenshot menu”.

The use case “Share Gameplay”, classified as essential, resulted from the “share menu” feature. This menu has four options: (1) “Upload Video”; (2) “Upload Screenshot”; (3) “Broadcast Gameplay”; (4) “Share Game Log”. It is presented by clicking the social button from the game options menu or the share menu shortcut. “Upload Video” is classified as essential and resulted from the “video clips menu” feature. It allows players to select and upload gameplay videos to social networks. Similarly, “Upload Screenshot” results from an “upload screenshot menu” feature. It allows the social network upload of player-selected screenshots. The “Share Game Log” use case resulted from the “share game log menu” feature and is dedicated to the social networks sharing of achievements, game stories and other information. The “Broadcast gameplay” use case, classified as conditional, resulted from the “broadcast gameplay menu” feature. It allows live social network broadcasting (streaming) of gameplays.

Our social overlay will also allow player to craft items or collections faster using social networks. Crafting is a method to create new game items or to complete collections, using different game elements that can be found in the game such as materials, pieces and cards. This feature resulted in “Craft” use case that was classified as a conditional. It enables players to send private messages asking missing game elements from their social network friends.

Finally, the passive party/team system, classified as an optional requirement, allows social network friends to act as passive members by performing actions even when not playing the game. These actions result in different game bonus that can be used to improve game-related experiences, e.g. get more attack for each friend that have the game installed. These bonus are optional and can be used to improve game-related experiences.

Among the different non-functional requirements of the social overlay we highlight the following: (1) support APO integration and full compatibility with Unity3D [21] for desktop games; (2) full Windows compatibility and when possible multiplatform support (Windows, Mac OS X and Linux).

4 System Design and Architecture

This section describes the design and architecture of the social overlay based on the requirements that were proposed.

Our system can be divided into three components. The first one is the desktop game client which includes the desktop game application and the social overlay tools. The second component is the web server. The web server includes the web services that receive and send information to the social overlay and the database that is used to store the information about the players’ requests. Lastly, the fourth component are the external social networks which include all the public media services. Fig. 3. shows the architecture of our system.

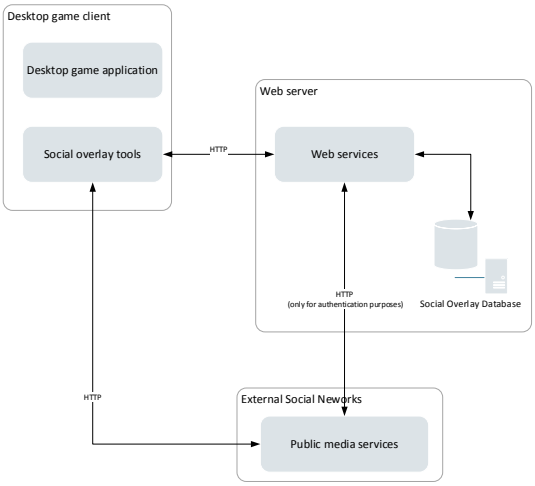


Fig. 3. System architecture for the social overlay integration in a desktop game.

4.1 Desktop game client

Desktop game client consists in two components: the desktop game application component, which is the part of desktop game client that contains the game's core logic and mechanics and the social overlay tools, which include all the social overlay features. To maintain some game styles attributes, the UI components or styles of each of these two components can be shared.

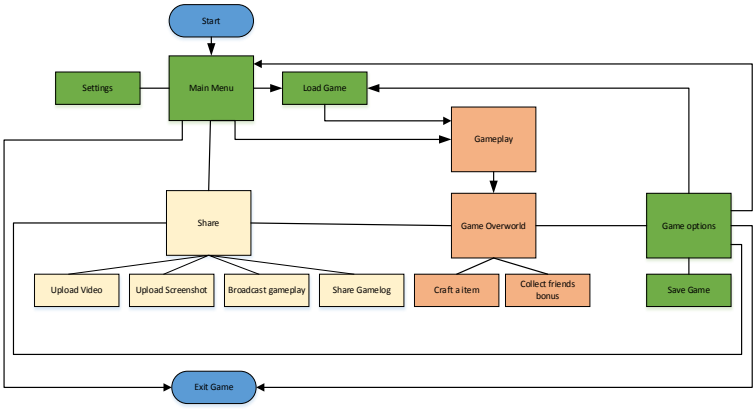


Fig. 4. Navigation chart/Game flow of social overlay integration with APO.

Fig. 4. presents a simplified navigation chart that includes integration of share, craft and passive party system tools with desktop game application. The first one includes a menu for the sharing of videos, screenshots and game-related stories and

for the broadcasting of the gameplay. Players can access to the share menu by clicking in the share option of the game menus or directly using the share menu hotkey. The second and third tools are only accessible inside the game overworld, which is the area that interconnects all the game locations, e.g. combat areas. With the craft, players can request help of social network friends' to create new inventory items and finally with passive party, players can collect social network friends' bonus to use game quests or fights. Additionally, social overlay allows the video record and screenshot capture in any part of the game using hotkeys or through the record and capture buttons. This game flow can be adapted in accordance to different game environments to create specific game tools or to customize the existing ones.

4.2 Social overlay tools

The social overlay tools include support for read and write data in the public media services (e.g. Graph API, Twitter API and Google API). In addition, the social overlay tools communicate with web services for grant permissions to players' social networks accounts and also to store and receive information about their social network requests. These tools are Unity3D Assets [24] for desktop games and are written in C#.

The choice of additional development technologies for the implementation of social overlay tools was made only based in the evaluation of the technical implementation specifications and in the verification of which tools responded better to the project needs. Screenshot Capture was implemented using the native function [26] of Unity3D for this effect. This function is compatible with Windows, Mac OSX and Linux. Regarding to video recording was concluded that the two best development options were: (1) the use of DirectShowNET [27]; (2) the use of FFMPEG [28]. As DirectShowNET is only available for Windows applications was decided to use the FFMPEG library, because it supports Windows, MAC OSX and Linux. This library was implemented with the help of Ruslan-B FFmpeg.AutoGen [29], which is an open-source wrapper for FFMPEG 2.5.2, compatible with Mono. In what regards to share, craft and passive party tools were analysed technologies for three types of social network integrations: Facebook integration, Twitter integration and Google integration. As these social overlay features are intimately related to the "external social networks" component, they will be presented in the next section.

4.3 External Social Networks

The social overlay tools, as well the web server are capable to interact with existing social networks. The credentials of the users are only used inside of social networks sites to acquire access tokens and are not stored. Our system includes support for the integration of Facebook's Graph API, Twitter REST API and Google APIs.

The **Facebook's Graph API** [30] is the low-level HTTP-based Application Programming Interface (API) of Facebook Platform that allow developers to read and write data into Facebook, to interact with it developers' applications need to use the OAuth 2.0 protocol [31] to obtain a Facebook access token for each user. Regarding

to Facebook integration was analysed a solution proposed by Paul Price [32]. This solution uses a web server to handles the tokens exchange with HTTP Requests and a Mono 2.6 [33] compatibility version of Facebook SDK C# [34] to access to Graph API. The main disadvantage of the solution provided by Paul Price is that every time Graph API methods are used a new access token is requested, even if it is not expired. To overcome this disadvantage was decided to implement a reformulation of Paul Price solution with the following changes: (1) The internet Connection is validate every time that the players do Facebook requests; (2) The login prompt and generation of access token are only requested the first time that players use Facebook in game; (3) The last Facebook session is stored between game sessions until user decides to logout or revokes application access; (4) Facebook session is validate before the user makes Facebook requests to check if a new access token need to be requested; (5) the wrapper of Facebook SDK C#, as well the web server implementation were reformulated according to technical implementation specifications.

The **Twitter REST API** [35] is the main application programming interface that allows developers to read and write Twitter platform data, to interact with it developers' applications need to use OAuth 1.0 protocol [36], similar to Facebook this protocol is used to acquire an access token for each user. As none of libraries analysed were compatible with mono 2.6, to build a more flexible integration of Twitter was decided to do a new implementation that combines both "Let's Twitter in Unity" asset [37] and Paul Price Facebook solution [32]. "Let's Twitter in Unity" is a free and open-source Unity asset that helps developers to deal with OAuth 1.0 protocol authentication of Twitter API 1.1. This asset only provides a pin-based authentication flow, so users have to manually insert a pin code in the game. To overcome this limitation our Twitter integration uses a web server and includes the following steps: (1) The internet Connection is validate every time that gamers do Twitter requests; (2) A HTTP POST calling a request token to the API is done. The login prompt and generation of request token are only requested the first time that players use Twitter in game; (3) default user browser is opened and he logins with Twitter account and accepts application permissions; (4) user is redirected to a web server page and success message is shown, meanwhile the OAuth verifier is passed back to the game; (5) the request token and OAuth verifier are converted to a access token; (6) The access token is used to do HTTP requests to the Twitter API; (7) OAuth tokens are stored until they are valid or until the user decides to do logout.

Google APIs [38] is a group of Application Programming Interfaces (APIs) provided by Google that allow the communication of third-party applications with Google services. Similar to Facebook, Google APIs use the OAuth 2.0 protocol for authentication and authorization. For the integration of Google services was decided to develop a new implementation that combine the use of Google APIs with HTTP Requests and Paul Price Facebook solution [32]. Similar to the solutions presented to Facebook and Twitter, this implementation contains the following steps: (1) The internet Connection is validate every time that gamers do Google requests; (2) A HTTP GET calling a request authorization token to the API is done. The login prompt and generation of request token are only requested the first time that players use Google account in game; (3) default user browser is opened and he logins with Google account and accepts application permissions; (4) user is redirected to a web

server page and success message is shown, meanwhile the authorization token obtained via login dialog is exchanged for a access token and a refresh token; (5) the desktop game polls the web server to check if access token for that user is already available; (6) The access token is used to do HTTP requests to Google APIs; (7) Access and refresh tokens are stored until they are valid or until the user decides to do logout. Contrary to Facebook and Twitter APIs, an additional access token verification step is done, because Google access tokens have a short live time and they need to be updated from time to time using a specific refresh token [39].

4.4 Web Server

The web server uses a basic ASP.NET MVC website and a MySQL database for the monitoring of players requests. In addition to the support of different HTTP request methods (e.g. GET, POST, PUT), the ASP.NET MVC application also includes a JSON library [40] to send and receive data information in this format.

The web server contains two different tiers: the user interface component (UI component), which is the part of the web server that include the views for the presentation of user interface layer to the players and the backend component, which is the part of the web server that provides the business and logic layer. The UI component provides the interface of the web browser instances in the desktop game client and also presents the basic information about the game and privacy policy. The backend component is the core of the web server and is responsible for ensuring the communication between the desktop game client and the other two components of the web server: the Database and the UI Component. Besides this, the backend component also include support for authentication and authorization of the desktop game client application in the social networks. Since this functionality is common to all social overlay tools the web server provides a simple and clear interface for each social network that can be used by all the modules of the social overlay.

The database is responsible for the persistence and monitoring of players' activity in social networks, for example, for obtain the statuses of social networks requests and for prevent duplicate social networks requests. This external database is also used as an information repository for the public players' information in social networks (name, gender, username and user ID).

5 Prototype Implementation

This prototype was developed in C# with Unity3D, additionally we used FFMPEG libraries for capturing system and a modified version of Facebook SDK C# for the connection with the Facebook. Our web application on the server is developed in ASP.NET and use MVC web application to ensure communication between the social overlay of the game client and the database and external social networks components.

The social overlay can be execute on Windows, Mac OSX and Linux, but we currently working in a compatibility version of the record feature for Mac OSX and Linux, because it requires different installations of FFMPEG libraries. Below is presented a compatibility table for each social overlay feature:

Table 2. Compatibility table of social overlay features.

Feature	Windows x86/x64	Mac OS X x86/x64	Linux x86/x64
UI components	√	√	√
Capture Screenshots	√	√	√
Record Videos	√	×	×
Facebook Share (upload videos, screenshots and stories)	√	√	√
Twitter Share (upload videos, screenshots and stories)	√	√	√
YouTube Share (upload videos)	√	√	√

As the prototype is in first stage only the essential requirements were fully implemented, this requirements will be presented in four sections: (1) Share Menu; (2) Record and Upload Videos; (3) Capture and Upload Screenshots; (4) Share Game Log.

5.1 Share Menu

To share videos, screenshots, gameplays and game-related stories, all the player needs is a Facebook, Twitter or Google account. The players have access to the following menu (Fig.5):

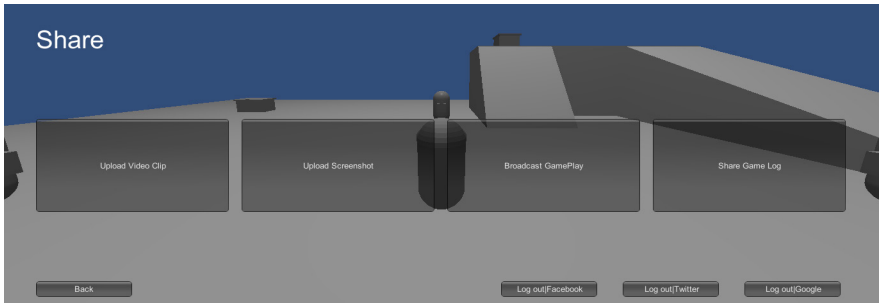


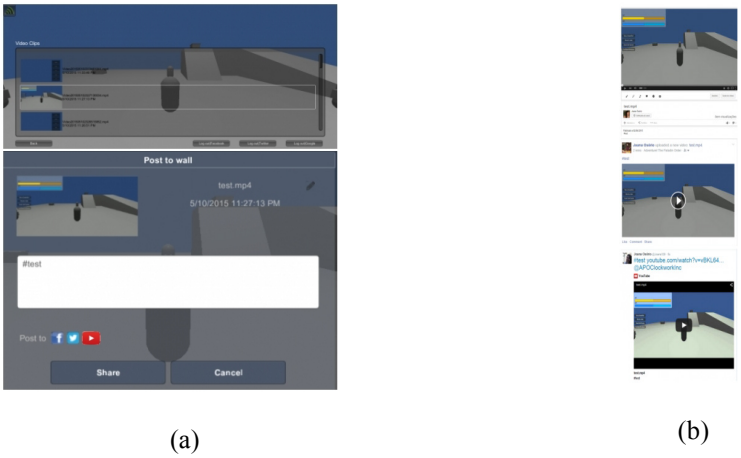
Fig. 5. Share menu of APO' social overlay

This menu includes "Upload Video Clip", "Upload Screenshot", "Broadcast Gameplay" and "Share Game Log" options, but currently the option "Broadcast Gameplay" is not enabled.

5.2 Record and Upload Videos

The players can record short video clips (3s~15 minutes) while playing a game and upload them to social networks. To record players have to press and hold the record button (or record shortcut) for at least 1 second.

To upload a video clip (Fig. 6) the player needs to: (1) Select a video; (2) Select a social network (Facebook, YouTube or Twitter or all the three); (3) (Optional step) Enter a comment; (4) (Optional step) Change video title; (5) Click in the share button.



(a) (b)

Fig. 6. Upload an APO' video to Facebook, Twitter and YouTube using the social overlay: (a) select a video to upload to social networks; (b) samples of posted videos in the social networks.

In the case of Twitter, as this plataform does not currently support the uploading of videos in desktop plataforms, users are asked to upload their videos to their youtube account and share the link to Twitter (Fig. 7). This option can be saved for next requests.

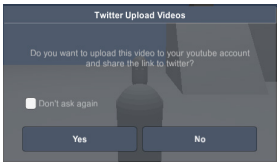
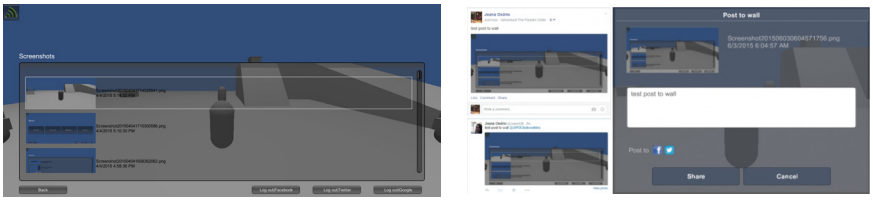


Fig. 7. User prompt to request permission for upload video to players' YouTube accounts and share the link in Twitter.

5.3 Capture and Upload Screenshots

The players can take screenshots while playing a game and upload them to social networks. To take a screenshot players have to press and hold Screenshot button (or screenshot shortcut) for at least 1 second.

To upload a screenshot (Fig. 8) the player needs to: (1) Select a screenshot; (2) Select a social network (Facebook, Twitter or both); (3) (Optional step) Enter a comment; (4) Click in the share button.

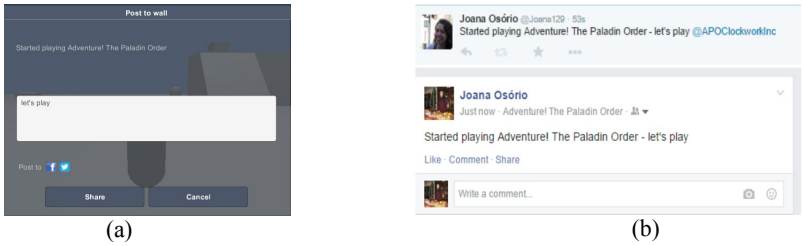


(a) (b)

Fig. 8. Upload an APO' screenshot to Facebook and Twitter using the social overlay: (a) Select a screenshot to upload to social networks; (b) Share the selected screenshot to Facebook and Twitter with message "test post to wall".

5.4 Share Game Log

Players can share information about trophies and game events on social networks. Game-related stories are captured and stored by the desktop game application automatically and them they are read by the social overlay game log tool. By default, social overlay game log tool already capture some common stories of the games such as "started playing a game" event. To share a story (Fig. 9) the user needs to: (1) Select a story; (2) Select a social network (e.g. Facebook, Twitter); (3) (Optional step) Enter a comment; (4) Click in the share button.



(a) (b)

Fig. 9. Upload story to Facebook and Twitter using the social overlay: (a) Share windows of game log option after select "Started playing" story; (b) "Started playing" story in Twitter and Facebook.

6 Alpha testing

The prototype of the social overlay based on the presented implementation, has undergone a preliminary testing phase that we called alpha testing. During this phase the social overlay among with Adventure! The Paladin Order (APO) game were evaluated with a survey using Google Forms, by a small number of players aged between 19 and 25 that are members of the APO's test team. The testers received a

link with game documentation and instructions. The testing was focused on the user needs and workflow desires for this kind of system more than on usability and graphics, and the users executed the tests by themselves without direct or indirect supervision. After testing they filled up the Google Form. Although the main purpose of the alpha testing was to identify the main limitations of our overlay, this testing also gave some insights to product awareness and future work.

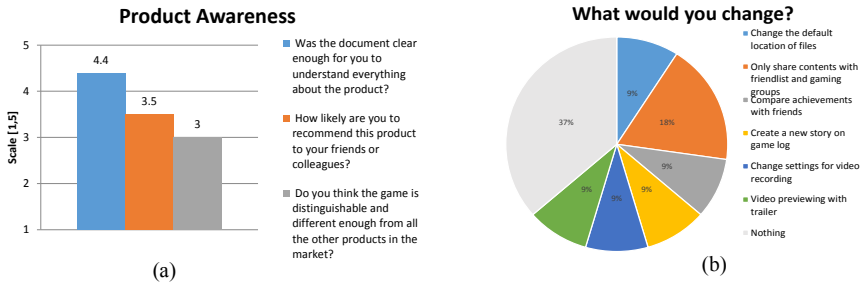


Fig. 10. APO' test team opinions: (a) Product awareness (scale of 1 to 5); (b) Product problems and limitations (open answer).

Regarding the product awareness (Fig. 10 (a)) testers found that the documentation and overall of the social overlay tools work well, however they did not consider our product distinguishable enough. We think that with the roadmap features planned we will manage to improve this aspect.

The major part of testers (30%, Fig. 10 (b)) did not report any limitations or problems of the features that are currently implemented, but some testers (20%, Fig. 10 (b)) do not like to post directly to their public social network wall and would rather post only to their friend list or gaming groups. A minor part of them (9%, Fig. 10 (b)) also want to change the default path where videos and screenshots are stored, compare achievements with friends and create personalized stories in game log. Finally, they request to change the settings for video recording, for example change the video frame rate (9%, Fig. 10 (b)) and to view a trailer of the video (9%, Fig. 10 (b)) instead of showing only a thumbnail. These limitations will be analysed and prioritized for future developments.

Our Clockwork Inc. team also suggested some roadmap features for the options of share menu that are already implemented, from which we highlight the following: (1) record the recent minutes of gameplay continuously and automatically recorded; (2) record the best moments of the game automatically; (3) capture screenshots of the best moments of the game automatically; (4) post to specific group/communities; (5) use private message to share screenshot; (6) use private message to share game log message. All these suggestions were classified by the testers with an average score of 3.7 or above in a scale of 1 to 5. Besides the roadmap features suggested by our team APO's test team would like to have: (1) support for Stream; (2) support for Skype; (3) support for specific game community such as Achievement hunter and Speedrunning community; (4) support for Reddit; (5) support for Vine.

7 Conclusion and Future Work

In this paper, we presented a prototype of a social overlay for desktop games that was tested and developed in the context of Adventure! The Paladin Order game, in order to provide social media resources for the personalization of gameplay experiences of the players.

This prototype is in its first stage (alpha version), therefore only the essential requirements were implemented and significant development is required to improve these requirements and implement the other three missing requirements: “Broadcast gameplay”, “Craft an item” and “Use passive party system”.

The next step of this work, during the beta phase implementation, is the evaluation and prioritization of the suggestions and problems reported by the test users. The beta testing phase will conduce to a bigger engagements of the APO’s test team in the game. All the suggestions, problems and bugs reported will be collected using online surveys and individual test sessions with screen sharing or alternatively with recorded videos, so we can observe and analyse the reactions and complains for each tester. In addition, regarding to the recording of videos and the sharing of videos on Twitter we need to improve some aspects related to the compatibility and usability, because currently players cannot record in Linux and Mac OSX systems and to share videos on Twitter the player’s Google account is needed. Although our service does not address anyone under the age of 13 ("Children"), our targets are players of different age groups, so we also have to take special care in the developing of the user interface which needs to be more player-friendly and predictable.

With this first alpha testing analysis we conclude that this social overlay will not only play an essential role in the integration of a social overlay for multiplatform desktop games (Windows, Linux and Mac OSX), but it will also contribute to the development of social media integration in games.

Acknowledgments. Our thanks go to Tiago Magalhães, creator of “Adventure! The Paladin Order” game and also for APO’s test team for all the help and dedication in the construction of this prototype. This work was supported by Instituto Superior de Engenharia do Porto (ISEP) and supervised by Professor Filipe Pacheco.

References

1. Konert, J., Göbel, S., Steinmetz, R.: Video Game Personalization via Social Media Participation. In: Göbel, S. and Wiemeyer, J. (eds.) Games for Training, Education, Health and Sports. pp. 35–46. Springer International Publishing (2014).
2. Hayati, P., Potdar, V., Talevski, A., Firoozeh, N., Sarenche, S., Yeganeh, E.A.: Definition of spam 2.0: New spamming boom. In: 2010 4th IEEE International Conference on Digital Ecosystems and Technologies (DEST). pp. 580–584 (2010).
3. Clockwork Inc.: Adventure! The Paladin Order, http://tdmei.azurewebsites.net/.
4. Duggan, M., Ellison, N.B., Lampe, C., Am, Lenhart, a, Madden, M.: Demographics of Key Social Networking Platforms, http://www.pewinternet.org/2015/01/09/demographics-of-key-social-networking-platforms-2/.

5. Shin, D.-H., Shin, Y.-J.: Why do people play social network games? Comput. Hum. Behav. 27, 852–861 (2011).
6. zyngaAdmin: FarmVille, https://company.zynga.com/games/farmville.
7. Hof, R.: Social Games: From the Farm to the Mafia, http://is.gd/PkAIZK, (2010).
8. Lynley, M.: Electronic Arts' The Sims Social hits 4.6 million daily players a week after launch, http://is.gd/U8K8UZ.
9. Andrew, W.: Half a billion people have installed "Candy Crush Saga," http://is.gd/H7qWtG, (2013).
10. Fjællingsdal, K.: Let's Graduate - A thematic analysis of the Let's Play phenomenon. (2014).
11. Pires, K., Simon, G.: YouTube live and Twitch: a tour of user-generated live streaming systems. In: Proceedings of the 6th ACM Multimedia Systems Conference. pp. 225–230. ACM (2015).
12. Brunelle, W.: Social Media and Video Games, http://is.gd/iNRL2K.
13. Mack, C.: Facebook and Twitter Go Live on Xbox 360, http://is.gd/Px19hr, (2009).
14. Kelion, L.: PlayStation 4 console announced by Sony in New York, http://www.bbc.com/news/technology-21526450, (2013).
15. Takahashi, D.: Overwolf launches social overlay for online games, http://is.gd/7Txi33, (2011).
16. Echobit LLC: Evolve, https://www.evolvehq.com/welcome.
17. Kietzmann, J.H., Hermkens, K., McCarthy, I.P., Silvestre, B.S.: Social media? Get serious! Understanding the functional building blocks of social media. Bus. Horiz. 54, 241–251 (2011).
18. Loguidice, B., Loguidice, C.: My Xbox: Xbox 360, Kinect, and Xbox LIVE. Que Publishing (2012).
19. Raby, M.: Microsoft retires Facebook, Twitter Xbox apps, http://is.gd/Fu4Vil, (2012).
20. Overwolf: Overwolf supported games, http://www.overwolf.com/supported-games/.
21. Unity: What is Unity?, https://unity3d.com/unity.
22. K900 (Product Owner of Evolve): Looking for Evolve, the game from Turtle Rock? Sorry, but you're in the wrong place, http://is.gd/samZVx, (2015).
23. Hurk, A.M. van den: Social Media Crisis Communications: Preparing for, Preventing, and Surviving a Public Relations #FAIL. Que Publishing (2013).
24. Unity: Submitting your content to the Asset Store, http://unity3d.com/asset-store/sell-assets.
25. Schell, J.: The Art of Game Design: A Book of Lenses, Second Edition. A K Peters/CRC Press, Boca Raton (2014).
26. Unity Technologies: Application.CaptureScreenshot, http://is.gd/Nx3NYM.
27. DirectShowNet: DirectShowNet library, http://directshownet.sourceforge.net/about.html.
28. Tomar, S.: Converting Video Formats with FFmpeg. Linux J. 2006, 10– (2006).
29. Balanukhin, R.: Ruslan-B/FFmpeg.AutoGen, http://is.gd/yR1ZFW.
30. Facebook: Graph API Overview, http://is.gd/RmaFCy.
31. Hardt, D.: The OAuth 2.0 authorization framework. (2012).
32. Price, P.: Connecting to Facebook from your Unity desktop game, http://www.ifc0nfig.com/accessing-facebook-unity-game-2/.
33. Mono: Release Notes Mono 2.6, http://is.gd/X3nm76.
34. Facebook SDK for .NET, http://csharpsdk.org.
35. Twitter: REST APIs, https://dev.twitter.com/rest/public.
36. Hammer-Lahav, E.: The oauth 1.0 protocol. (2010).
37. Yang, Y.: Let's Tweet In Unity, https://www.assetstore.unity3d.com/en/#!/content/536.
38. Google: Google APIs Explorer, https://developers.google.com/apis-explorer/#p/.
39. Google: Using OAuth 2.0 for Web Server Applications offline access, http://is.gd/umx0rg.
40. Shrestha, P.: SimpleJson, https://github.com/facebook-csharp-sdk/simple-json.

Como separar o trigo do joio?

Ou: Como selecionar a melhor fotografia de um conjunto de fotografias semelhantes

André Alves, Fernando P. Birra, and João M. Lourenço

NOVA Laboratory for Computer Science and Informatics
Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade NOVA de Lisboa, Portugal

Resumo O advento da fotografia digital está na base de uma clara mudança de paradigma no processo de gestão da fotografia por amadores. Porque tirar mais uma fotografia agora não representa qualquer custo adicional, é frequente tirarem-se múltiplas fotografias ao mesmo sujeito, na expectativa de que uma delas corresponda aos padrões de qualidade desejados, em termos de iluminação, foco e enquadramento. Assumindo que a questão do enquadramento se resolve facilmente recorrendo ao recorte (*crop*) da fotografia, tem-se ainda assim que selecionar qual das várias fotografias bem enquadradas, tecnicamente parecidas em termos de iluminação e foco, vamos guardar (e por oposição quais vamos descartar). A escolha da melhor fotografia com base na observação visual em ecrã de computador é um processo muito pouco preciso e, portanto, gerador de sensações de insegurança que resultam, muitas vezes, na opção de não descartar nenhuma das várias fotografias semelhantes. Neste artigo propomo-nos endereçar a questão de como ajudar um fotógrafo amador a selecionar a melhor fotografia de um conjunto de fotografias semelhantes em termos técnicos (foco e iluminação) e de enquadramento. Este processo é baseado num *workflow* suportado por um pacote de *software*, que com alguma ajuda do utilizador permite ordenar um conjunto de fotografias semelhantes, sendo assim possível escolher aquela que melhor corresponde às expectativas e dando segurança e conforto na eventual eliminação das restantes.

1 Introdução

O registo da luz de uma cena por parte do sensor de uma câmara digital, como por exemplo uma *Digital Single-Lens Reflex*, pode ter resultados bastante diferentes consoante os parâmetros definidos na câmara, sendo assim necessário um certo grau de perícia para dominar este processo. Desta forma, com o objetivo de aumentar a probabilidade de se obter o resultado pretendido, é prática comum entre os fotógrafos amadores tirar várias fotos ao mesmo motivo e em modo automático, na esperança de que uma delas satisfaça os seus requisitos.

Algumas das fotos capturadas podem não corresponder às expectativas. Uma *focagem imprópria*, possivelmente causada por uma má identificação do ponto

de focagem ou por uma má definição da profundidade de campo, pode resultar numa fotografia em que o motivo se encontra desfocado, contribuindo assim para o descontentamento do fotógrafo. Por outro lado, caso a câmara se mova durante o momento da exposição, o resultado será uma fotografia que contém uma *desfocagem geral*, sem qualquer claro centro de interesse. Ainda assim, a desfocagem pode ser causada propositadamente pelo utilizador com o objetivo de dar a impressão de *movimento na cena*. Dado que este efeito requer um certo grau de perícia, nem sempre poderá ser possível obtê-lo numa única tentativa, resultando em várias fotos muito parecidas. A sensibilidade do sensor da câmara perante a luz existente resultará em fotografias com *diferentes níveis de ruído*, sendo que o ruído pode ser subjetivamente considerado um fator degradante para a qualidade da fotografia. No aspeto da cor, esta tem um papel integral não só na percepção visual mas também nas emoções que uma imagem cria no espetador. Para se obter a exposição ideal, o montante correto de luz deve ser capturado. Especialmente quando usa o modo automático da câmara pode ocorrer que o fotógrafo capture imagens com *diferentes níveis de exposição*. Uma captura excessiva de luz dá origem a uma fotografia sobre-exposta, dominada por tons claros. Por outro lado uma captura escassa de luz dá origem a uma fotografia sub-exposta, dominada por tons escuros. Ainda na questão da cor, tem-se que as câmaras digitais tendem a capturar a luz com as suas características primitivas, podendo assim capturar cores que fogem à percepção do sistema visual humano. Como resultado tem-se fotografias com um *elenco de cores irrealista*, normalmente chamadas de fotografias quentes, predominadas por cores avermelhadas ou fotografias frias, predominadas por cores azuladas.

Para além de todos os desafios que o fotógrafo enfrenta no momento da captura da fotografia, este tem de lidar com outros problemas adicionais. O grande desafio ocorre na realidade quando, mais tarde, o fotógrafo transfere o conjunto de fotos semelhantes para o computador na expectativa de escolher a que considera melhor e se sente frustrado pela incapacidade de escolher com segurança quais as fotos a preservar e quais eliminar. É neste processo que este artigo se foca. Os principais desafios na escolha da melhor foto são: como lidar com o *tamanho da amostra*, podendo o número de fotos variar entre um par e dezenas; como lidar com a lentidão resultante de manipular *ficheiros de grande dimensão*; como lidar com o facto de que a *resolução* de uma fotografia tirada com uma DSLR ser normalmente muito maior do que a resolução de um ecrã de computador, obrigando à redução do tamanho da imagem com consequências na qualidade da mesma, ou a um processo de análise de apenas partes da imagem. Desta forma o processo de escolha da melhor foto requer necessariamente que se navegue pelo universo de fotos existentes, podendo-se tornar um processo demorado, tedioso e propenso ao erro, gerador portanto de insegurança e frustração.

Neste artigo apresentamos uma proposta de uma metodologia, acompanhada por uma ferramenta de *software* inovadora, que visa ajudar o utilizar no processo de seleção da melhor foto entre um conjunto de fotos semelhantes. Para tal é feita uma análise aos problemas anteriormente referidos (focagem, desfocagem, estimativa de ruído e análise da cor) em fases separadas, sendo que o utiliza-

dor pode ir sucessivamente rejeitando as fotografias que não preenchem os seus requisitos, até ficar com uma amostra bastante reduzida, tornando assim este processo numa experiência gratificante. A implementação da análise computacional integrada na ferramenta de apoio ao *workflow* de seleção é baseada num conjunto de algoritmos já disponibilizados pelo OpenCV [6].

Este artigo está organizado da seguinte forma: na Secção 2 é apresentado o trabalho relacionado. Na Secção 3 é apresentado o procedimento idealizado para ajudar o utilizador a escolher a melhor foto, nomeadamente o *workflow* que o compõe bem como os desafios levantados por cada uma da suas fases. Na Secção 4 é apresentada a metodologia utilizada para implementação do *workflow*. Na Secção 5 é apresentada uma avaliação sobre os métodos já implementados, sendo assim possível analisar o seu comportamento e antever o impacto da sua utilização pelo utilizador final. Por fim, na Secção 6 é comentado o trabalho realizado e exposto o trabalho futuro.

2 Trabalho Relacionado

Atualmente existem várias técnicas propostas para a análise de imagens segundo diferentes parâmetros, nomeadamente deteção de *focus* [12,9], deteção de *motion blur* [10,16] e deteção de ruído [8,14]. No entanto, segundo as pesquisas realizadas, continua a haver uma escassez de trabalhos que combinem estes métodos com o objetivo de selecionar a melhor foto de um conjunto de imagens semelhantes. Esta secção apresenta e compara dois trabalhos cujo o âmbito está relacionado com o trabalho proposto neste artigo.

2.1 Automatic Photo Selection for Media and Entertainment Applications

Potapova et. al. [15] propuseram um método para a seleção das melhores fotos de um álbum. Numa primeira fase são identificadas fotos de baixa qualidade que não são tidas em conta nos processamentos posteriores. Fotos afetadas por artefactos de compressão, que segundo os autores se assemelham a ruído, são detetadas com recurso a um filtro de *deblocking* e *deriging* [4]. Para ajuste dos parâmetros do filtro é realizada uma análise sobre a tabela de quantização q, de tamanho $3 * 3$, do canal de brilho: $k = 1/9 \sum_{i,j=1}^{3} q_i, j$.

Segundo as experiências dos autores, caso k tenha um valor superior a 6,5 então a fotografia é fortemente afetada por artefactos de compressão. Imagens de baixo contraste, particularmente afetadas por *backlighting*, são detetadas com recurso ao histograma de brilho da imagem, tendo em conta informação como os valores de tons em sombras, tons médios e o tom máximo do histograma. Estes valores servem como parâmetros para o algoritmo de aprendizagem automática AdaBoost [5] de forma a distinguir imagens de baixo contraste. Para a identificação de imagens desfocadas, cada imagem é convolvida várias vezes com filtros *high-pass* de vários tamanhos sendo que a entropia da variação dos histogramas nas arestas caracteriza o achatamento ou o pico do histograma. Para além disso

é também analisada a diferença entre a versão original da imagem e uma versão suavizada. Caso a diferença seja não seja significativa então a imagem original possivelmente está desfocada.

Após a remoção das fotografias de baixa qualidade, o próximo passo é agrupar as restantes. Para tal é tida em conta a câmara com a qual a foto foi tirada e o momento em que foi tirada. Esta informação é obtida a partir da informação EXIF [7]. Posto isto, os autores assumem que a foto mais apelativa e relevante de cada grupo é a mais saliente. Desta forma, para cada foto é construído um mapa de saliência com base nos mapas de intensidade, orientação e cor, sendo atribuído a cada um destes mapas um peso específico.

Os autores afirmam obter resultados bastante positivos com este método. No entanto não nos foi possível testá-lo visto que não encontrámos nenhum produto de *software* com a sua implementação. O método de deteção de fotos desfocadas poderá vir a ser interessante no contexto deste trabalho, possivelmente como um pré-processamento para a deteção de *motion blur*. Pelo contrário, uma fotografia tirada a uma cena cujos objetos tenham cores semelhantes será uma fotografia de baixo contraste, o que não implica necessariamente que seja indesejável. Desta forma, no contexto deste trabalho esse método não será viável. Em relação à deteção de artefactos de compressão, fica em aberto uma análise mais profunda sobre este método. No entanto a nossa primeira opção passará pela tentativa de implementação de um método que permita ter uma estimativa do nível de ruído na imagem.

2.2 Tiling Slideshow

Chu et. al. [2] propuseram um método para gerar *slideshows* audiovisuais em que várias fotos com características semelhantes são exibidas na mesma *frame*. Embora este método não tenha o propósito final de selecionar a melhor foto, torna-se interessante referenciá-lo visto que um dos procedimentos consta em remover fotos de baixa qualidade. Os autores consideram fotos desfocadas, sub-expostas e sobre-expostas como tendo baixa qualidade. Para a deteção de fotos de baixa qualidade, foi adotado um método baseado em *wavelets* que analisa as características das arestas em diferentes resoluções. Fotos sub-expostas e sobre-expostas são detetadas através do cálculo do número de *pixels* escuros e claros em cada foto. Caso estes valores superem um determinado *threshold* as fotos serão consideradas sub-expostas (*pixels* escuros) ou sobre-expostas (*pixels* claros). Mais uma vez, o facto de uma foto ter baixo contraste não implica necessariamente que tenha baixa qualidade pelo que seguiremos uma abordagem diferente para a deteção de fotografias com um nível de exposição inadequado.

3 Procedimento para escolha da melhor fotografia

O procedimento proposto neste artigo para a escolha da melhor foto tem como base um *workflow*, pelo qual o utilizador é guiado. O utilizador deverá incorporar o resultado de cada fase do *workflow* no seu processo de eliminação seletiva das

fotografias (semelhantes) que mais se afastam dos seus objetivos e expetativas, chegando no final a um conjunto reduzido de fotografias.

3.1 O Workflow

O *workflow* referente a este procedimento é composto por cinco fases, podendo ser analisado na Figura 1.

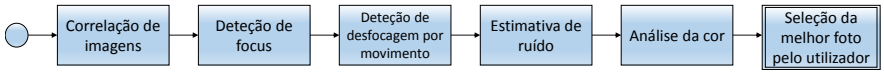


Figura 1: *Workflow* proposto.

A ordem definida para as fases do *workflow* reflete a relevância dos problemas por elas endereçados. Por exemplo, uma má focagem é um fator claramente eliminatório visto que é irreversível, pelo que estas fases se encontram logo no início do *workflow* (apenas precedido pela correlação das imagens que é um pré-requisito para o processamento das mesmas). É possível atenuar ou mesmo eliminar o efeito do ruído nas fotografias, às custas de alguma redução na qualidade da imagem, razão pela qual se considerou a análise do ruído como o terceiro problema mais relevante. Existem técnicas com impacto muito reduzido na qualidade da imagem que permitem alterar o nível de exposição e a temperatura de uma imagem, pelo que este é o último parâmetro a ser analisado no *workflow* proposto. Por fim, cabe ao utilizador escolher a melhor fotografia do conjunto restante.

Correlação de imagens: Com o objetivo de fazer uma análise comparativa entre as imagens do conjunto é necessário correlacionar as mesmas. O resultado deste procedimento permite identificar automaticamente pontos, ou áreas, equivalentes nas duas imagens. Esta fase não requer *input* do utilizador, tendo como propósito possibilitar uma análise comparativa entre imagens nas fases posteriores. Sendo natural que as imagens do conjunto variem em fatores como a profundidade de campo, ruído, distância focal ou orientação, o principal desafio desta fase consta em encontrar-se um método que seja invariante aos fatores mencionados, resultando assim numa correlação de imagens com um grau de precisão elevado. Visto que o conjunto de imagens a processar pode ser vasto, o processo de correlação tem de apresentar reduzidos tempos de processamento. Abordagens possíveis incluem correlacionar todas as imagens entre si, ou utilizar uma imagem de referência e correlacionar as restantes apenas com esta.

Análise das regiões focadas: Nesta fase, o objetivo é apresentar ao utilizador uma máscara sobre as regiões focadas de cada imagem, podendo de seguida o

utilizador eliminar as fotos que não apresentam os motivos de interesse devidamente focados. O principal desafio desta fase prende-se com a apresentação intuitiva, eficiente e precisa da máscara sobre as regiões focadas de cada imagem.

Detecção de desfocagem por movimento: Nesta fase é também apresentada uma máscara sobre as zonas das fotografias que têm uma desfocagem por movimento. No caso de a desfocagem ser resultado de movimento da câmara então a máscara cobrirá toda a superfície da fotografia. Caberá depois ao utilizador decidir se pretende fotografias com desfocagem por movimento ou não, eliminando as que não são pretendidas. A distinção entre desfocagem causada pelo motivo estar fora da profundidade de campo e desfocagem por movimento na cena será o principal desafio a ter em conta nesta fase.

Estimativa de ruído: Nesta fase será apresentado ao utilizador uma estimativa do nível de ruído existente em cada fotografia, podendo o utilizador eliminar as fotografias com um nível de ruído diferente do pretendido. O ruído pode ser visto como um conjunto de pontos alienados, cuja intensidade e cor não estão relacionados com os restantes *pixels*, estando espalhados ao longo da superfície da imagem. A deteção destes pontos ruidosos e consequente estimativa do nível de ruído existente em cada imagem define o principal desafio desta fase.

Análise da cor: Por último, esta fase servirá para distinguir e apresentar ao utilizador o nível de exposição de cada fotografia bem como a temperatura que esta contém. Uma análise correta dos histogramas de tons da cor da imagem poderá servir como ponto de partida para implementar com sucesso esta fase.

Seleção da melhor foto: Após a conclusão dos procedimentos que envolvem uma análise computacional, espera-se que a amostra final de fotografias seja bastante reduzida quando em comparação com a amostra inicial. Das fotografias restantes (no caso de haver mais que uma), cabe ao utilizador escolher a melhor com base no enquadramento e em parâmetros de composição.

4 Desenho do sistema

Tendo descrito cada fase do *workflow* proposto, esta secção tem como propósito apresentar a forma como se procedeu para resolver os desafios identificados na secção anterior. Até ao momento de escrita deste artigo foram implementados os procedimentos de correlação de imagens e de deteção de *focus*. Desta forma, o trabalho descrito incide maioritariamente sobre estes dois procedimentos. Para os restantes procedimentos é apresentado de forma sucinta algum *background* e algumas ideias de como abordar essa fase.

4.1 Correlação de imagens

Uma visão global da implementação seguida para este procedimento pode ser vista na Figura 2.

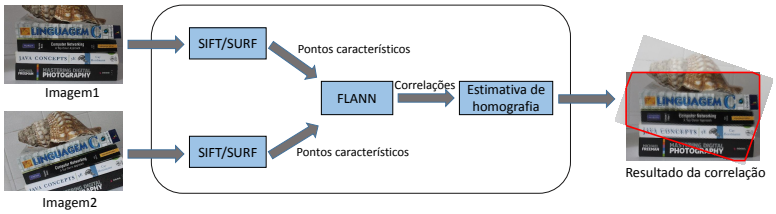


Figura 2: Visão global do procedimento de correlação de imagens.

Os algoritmos SURF [1] e SIFT [11] têm o propósito de detetar pontos característicos, ou seja, pontos distinguíveis em imagens. Ambos têm procedimentos semelhantes: numa primeira fase são detetados pontos característicos em cada imagem. De seguida, é calculado um descritor para cada um destes pontos. Este descritor contém principalmente a orientação e a escala em que o ponto foi detetado. Desta forma estes algoritmos tendem a ser invariantes a mudanças de escala e orientação mas apenas parcialmente invariantes a mudanças de iluminação.

Visto que as imagens podem variar significativamente em termos de iluminação, o nosso primeiro passo constou em normalizar as imagens de forma a torná-las o mais semelhantes possível. Para tal foi utilizada a operação de *Histogram equalization* disponível no OpenCV [6]. A Figura 3 demonstra o resultado da aplicação desta operação sobre uma imagem sub-exposta, sendo que o número de pontos característicos detetados é claramente superior após a operação de normalização. Na mesma figura é também possível ver que o número de pontos característicos detetados, após a operação de normalização, se aproxima do número de pontos característicos detetados numa imagem com a exposição ideal (referência).

Tendo sido calculados os descritores de cada ponto, o próximo passo será emparelhar pontos de interesse respetivos entre imagens diferentes. A biblioteca FLANN [13] contém um conjunto de algoritmos de *neighbour searches*, sendo capaz de escolher o algoritmo mais apropriado consoante o tipo de dados que recebe como entrada. Desta forma é possível encontrar o descritor, numa imagem, que é mais próximo daquele que o FLANN recebe como *input*. Para tal foram testados os métodos `match` e `knnMatch`. O método `match` devolve o descritor do ponto de uma imagem que é mais próximo ao ponto da imagem de referência. O método `knnMatch` devolve os k descritores mais próximos. O principal desafio desta fase consta em minimizar o número de falsos positivos, isto é, correlações de pontos não respetivos. Usando o método `match`, definiu-se um *threshold* global. Para todas as correlações entre pontos foi calculada aquela que tem a menor

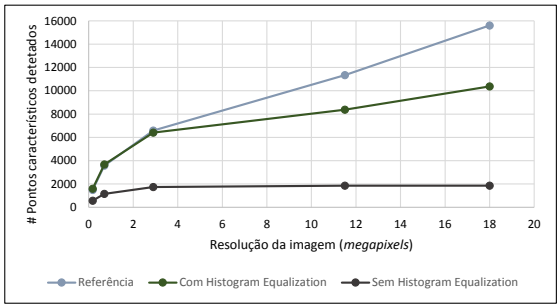


Figura 3: Impacto do *Histogram Equalization* na deteção de pontos característicos.

distância (min_dist). De seguida, para cada uma das restantes distâncias ($dist_i$), caso esse valor esteja abaixo de um *threshold* t ($dist_i < 2 * min\_dist$) a correlação é considerada como positiva, caso contrário é negativa. No entanto, tal como apontado por Lowe [11] a definição de um *threshold* global pode não ser a opção mais eficaz visto que alguns descritores são mais discriminatórios que outros. Desta forma Lowe [11] propôs um *ratio test* em que caso o rácio da distância entre o vizinho mais próximo ($v1$) e o segundo vizinho mais próximo ($v2$) seja superior a $1/1,5$ ($v1/v2 > 1,5$) então a correlação é considerada como positiva. Caso contrário é considerada negativa. Este teste permite acima de tudo a não eliminação de *matches* corretos como se pode comprovar pela Figura 4.

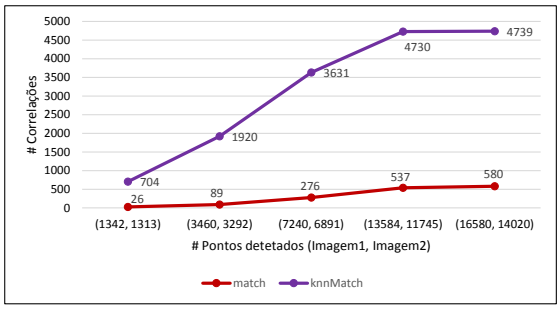
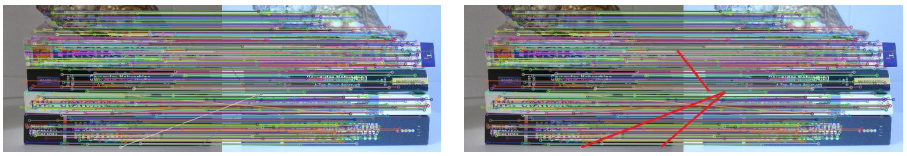


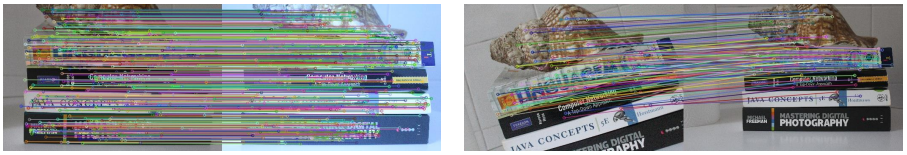
Figura 4: Comparação entre `match` e `knnMatch` no processo de correlação.

De forma a tornar este processo ainda mais eficaz e eliminar eventuais falsos positivos (marcados a vermelho na Figura 5) foi também calculada a matriz de homografia entre as imagens usando o método *random sample consensus* (RANSAC) [3]. O resultado final pode ser visto nas Figuras 6a e 6b.



(a) Pontos correlacionados. (b) Três falsos positivos.

Figura 5: Resultado da aplicação do FLANN.



(a) (b)

Figura 6: Resultado final da correlação de imagens após cálculo da matriz de homografia.

De forma a correlacionar todo o conjunto de imagens foi decidido que haverá uma imagem de referência, sobre a qual o utilizador trabalhará, sendo que apenas esta imagem será correlacionada com as restantes. Esta abordagem oferece vantagens acima de tudo em termos de eficiência visto que como exemplo, um caso ilustrativo de dez imagens, implica nove correlações. Já correlacionando todas as imagens entre si implicaria 45 correlações.

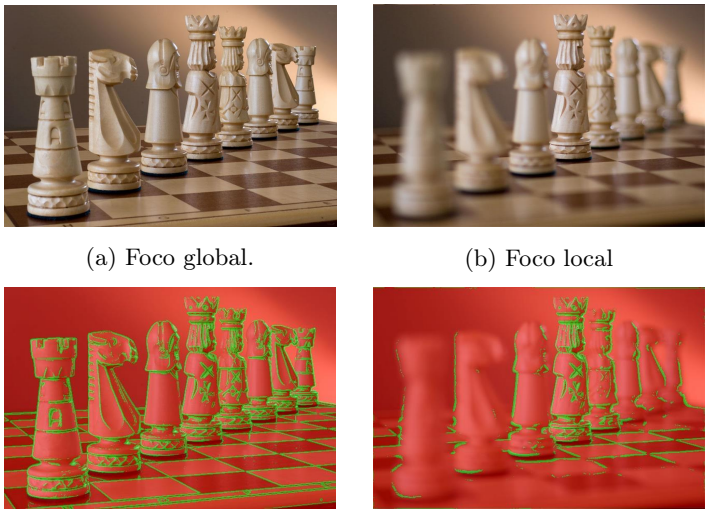
4.2 Deteção de *focus*

De forma a analisar as imagens em termos de focagem foram implementadas duas abordagens: a primeira define-se como uma análise global da focagem enquanto que a segunda é uma análise sobre uma região de interesse definida pelo utilizador.

O gradiente da imagem pode ser uma característica importante nesta análise visto que a intensidade da cor de uma imagem focada tende a ser maior do que a de uma imagem desfocada. Desta forma, a metodologia seguida para analisar as imagens em termos de focagem envolve o cálculo do gradiente, nomeadamente usando o filtro Sobel disponível no OpenCV [6].

A **análise global da focagem** tem como objetivo apresentar uma forma rápida e intuitiva de detectar e distinguir regiões focadas nas várias imagens. O algoritmo Sobel é aplicado sobre a versão em tons de cinzento de cada imagem, sendo que as regiões de notória variação de intensidade (ex.: arestas) são detetadas no *output*.

[1] Figuras 7a e 7b tiradas de: http://teachers.sduhsd.net/delliott/Files-Photo/deep%20vs%20shallow%20DOFchess%20pieces.JPG



(a) Foco global. (b) Foco local

(c) Máscara sobre Figura 7a (d) Máscara sobre Figura 7b

Figura 7: *Pixels* verdes têm uma intensidade superior ao valor do *slider*[1].

Os valores resultantes do algoritmo Sobel são depois normalizados entre 0 e 255 de forma a ser possível oferecer ao utilizador um *slider* que representa o valor da intensidade. Aos *pixels* cujo valor da intensidade é maior que o valor do *slider* é sobreposta uma máscara verde, aos restante é sobreposta uma máscara vermelha. O efeito pode ser verificado na Figura 7.

Pode ocorrer que as imagens do conjunto sejam bastante semelhantes em termos de focagem, o que torna a abordagem anterior ineficaz. Desta forma é também possível uma **análise sobre uma região de interesse** definida pelo utilizador na imagem de referência. As respetivas regiões nas restantes imagens são calculadas automaticamente via homografia. A cada região de interesse é aplicado o algoritmo Sobel, sendo de seguida calculada a média do valor da intensidade nessa região. A região cujo valor médio de intensidade for mais elevado é tida como sendo a mais focada.

4.3 Deteção de desfocagem por movimento

O gradiente da imagem define mudanças direcionais em termos de intensidade ou cor. Este pode ser visto como um vetor 2D que aponta na direção de maior variação, representando a sua magnitude a taxa dessa variação.

O método proposto em [10] visa distinguir desfocagem causada pelo motivo estar fora da profundidade de campo (*out-of-focus*) e desfocagem causada por movimento na cena (*motion blur*). A metodologia seguida consta em criar uma versão suavizada da imagem, dividida em várias regiões, tendo em conta que:

1. Se uma imagem contiver *motion blur*, as arestas com gradiente perpendicular à direção de movimento não serão suavizadas, tendo o gradiente maior magnitude apenas ao longo de uma direção.
2. Se uma região da imagem estiver desfocada por *out-of-focus*, a magnitude do gradiente é atenuada em todas as direções.

Com base nestas afirmações é construído um histograma direcional para todas as regiões da imagem. Cada entrada do histograma representa uma direção específica sendo o seu valor o número de *pixels* cujo gradiente tem essa direção. Desta forma, quando em face de *motion blur*, o histograma apresenta um pico distintivo na direção de movimento.

O método proposto em [16] é baseado num modelo de duas camadas em que cada pixel I é visto como uma combinação linear de cores de *foreground*, F, e cores de *background*, B, sendo α o peso de cada componente. Este modelo é definido da seguinte forma: $I = \alpha F + (1 - \alpha)B$, onde α pode ter qualquer valor no intervalo $[0, 1]$. Em imagens desfocadas, as cores de *foreground* e as cores de *background* tendem a misturar-se, nomeadamente nas arestas, tomando α valores decimais. Desta forma, é realizada uma análise sobre o modelo do canal α, definido por $\bigtriangledown\alpha.b \in \{-1, 1\}$, onde b é um vetor que define a medida de desfocagem na direção vertical e horizontal. Para regiões afetadas por *motion blur*, b tende a ser direcional, logo $\bigtriangledown\alpha$ é representado por linhas, enquanto que em situações de *out-of-focus*, $\bigtriangledown\alpha$ é espalhado por todas as direções, com um efeito radial.

4.4 Deteção de ruído

O método filtered-based [14] tem como ideia base obter-se uma versão suavizada da imagem ruidosa e de seguida subtrair essa versão suavizada à versão original. Isto porque os filtros de suavização tendem a atenuar ou eliminar o ruído existente sendo que a diferença entre versões irá fazer sobressair o ruído.

No método *block-based* [8] a imagem é dividida em vários blocos sendo que o desvio padrão de intensidade do bloco mais suave é considerado como sendo o valor do ruído existente. Nesta abordagem a obtenção de um bloco totalmente suave na imagem poderá ser realizada automaticamente ou com recurso a *input* do utilizador. No caso ilustrativo da Figura 8 encontram-se três regiões homogéneas com diferentes níveis de ruído.

O desvio padrão de intensidade da cor nas Figuras 8a, 8b e 8c é 14.8, 37.8 e 60.3, pelo que estes valores podem servir como um indicador do nível de ruído na imagem.

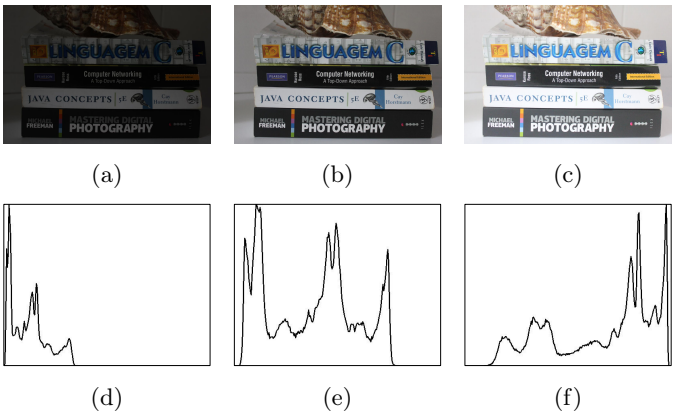
4.5 Análise da cor

A luminosidade da cor numa imagem pode definir o seu nível de exposição. Uma imagem sub-exposta é dominada por tons escuros, uma imagem sobre-exposta é dominada por tons claros enquanto que uma imagem com um nível de exposição balanceado contém os seus tons de cor espalhados por toda a escala.

(a) (b) (c)

Figura 8: Diferentes níveis de ruído.

Três imagens com diferentes níveis de exposição e os seus respetivos histogramas com a distribuição dos tons de cinzento podem ser vistos na Figura 9.



(a) (b) (c)

(d) (e) (f)

Figura 9: Diferentes níveis de exposição e respetivos histogramas.

As funções que definem os histogramas podem ser relacionadas entre elas. Uma análise comparativa entre histogramas permitirá, em princípio, ordenar as imagens em termos de nível de exposição.

5 Avaliação

Nesta secção são apresentados o protótipo e o desempenho das duas primeiras fases do *workflow* proposto. Todas as experiências foram realizadas num PC Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz 2.40GHz com 8 GB RAM a correr sobre o sistema operativo Windows 8.1 com 64 bits. As imagens utilizadas para realizar as experiências encontram-se disponíveis em https://github.com/apalves/Sets-of-similar-photos.

5.1 Protótipo

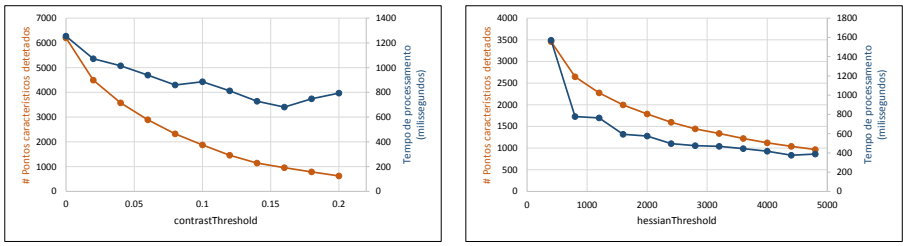
O desenvolvimento da ferramenta de *software* proposta tem tido como base um conjunto de algoritmos disponibilizados pelo OpenCV [6] usando a linguagem de programação C++. A unidade *highgui* do OpenCV tem sido utilizada como interface gráfica sendo que pela altura da escrita deste artigo a interface gráfica do utilizador ainda não se encontrava implementada.

5.2 Impacto de `contrastThreshold` no tempo de processamento e na deteção de pontos

O método `SIFT` é definido da seguinte forma:

```
SIFT::SIFT(int nfeatures=0, int nOctaveLayers=3, double
    contrastThreshold=0.04, double edgeThreshold=10, double sigma=1.6)
```

O parâmetro `contrastThreshold` é usado para filtrar pontos dúbios em regiões de baixo contraste. Desta forma, tendo em conta que as imagens podem variar significativamente em termos de iluminação, foram corridos vários testes variando o valor deste parâmetro. O impacto desta variação na deteção de pontos característicos e no tempo de processamento pode ser visto na Figura 10a. Os testes foram realizados com uma resolução de aproximadamente 0,7 *megapixels*.



(a) *contrastThreshold*. (b) *hessianThreshold*.

Figura 10: Impacto de *contrastThreshold* e *hessianThreshold* na deteção de pontos característicos e no tempo de processamento.

Naturalmente, há medida que o valor do `contrastThreshold` aumenta menos pontos são detetados sendo que o tempo de processamento também diminui variando entre os 750 e aproximadamente 1200 milissegundos.

5.3 Impacto de `hessianThreshold` no tempo de processamento e na deteção de pontos

O método `SURF` é definido da seguinte forma:

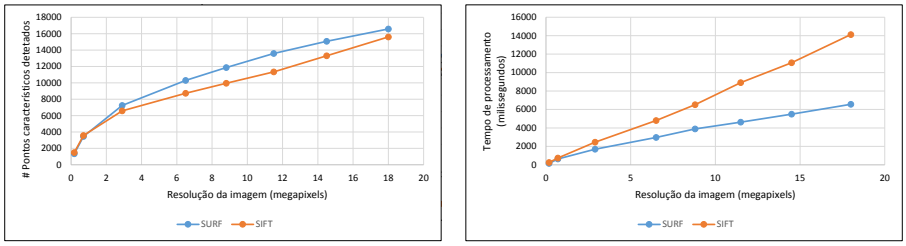
```
SURF::SURF(double hessianThreshold, int nOctaves=4, int nOctaveLayers=2,
    bool extended=true, bool upright=false )
```

Seguindo a mesma lógica de raciocínio foram realizados vários testes variando o valor do parâmetro `hessianThreshold`. O impacto desta variação pode ser visto na Figura 10b.

Comparando `SIFT` e `SURF` tem-se que o algoritmo `SURF` é mais rápido sendo que ambos os algoritmos apresentam um número significativamente elevado de pontos característicos detetados.

5.4 Impacto da resolução da imagem no tempo de processamento e na deteção de pontos

O impacto da resolução da imagem na deteção de pontos característicos e no tempo de processamento pode ser visto na Figura 11.



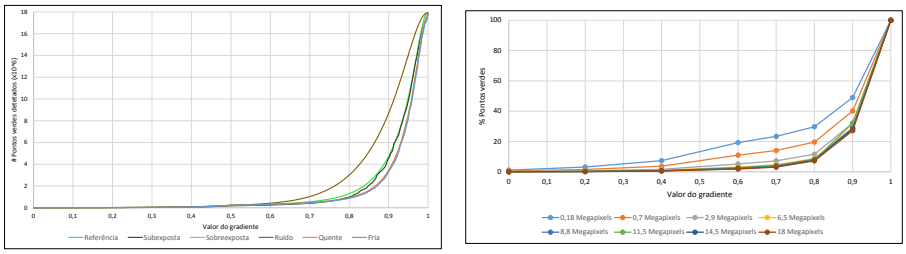
(a) Deteção de pontos característicos. (b) Tempo de processamento.

Figura 11: Impacto da resolução da imagem.

O número de pontos característicos detectados tende a ter a mesma ordem de grandeza usando o `SIFT` ou o `SURF` nas várias resoluções. A principal razão do `SIFT` apresentar tempos de processamento mais lentos prende-se com o facto de este processar as imagens em várias escalas de forma a que o resultado se torne invariante a mudanças de escala. Já o algoritmo `SURF` mantém sempre a mesma resolução da imagem, variando apenas o tamanho do *box filter* que é aplicado à imagem integral. Tendo em conta que a ferramenta proposta é interativa, torna-se importante ter tempos de processamento baixos com o intuito de melhorar a experiência do utilizador. Desta forma, a nossa opção passará, ao que tudo indica, pela utilização do algoritmo `SURF`.

5.5 Impacto das características da imagem no cálculo do gradiente

O impacto das características da imagem (ruído, nível de exposição, temperatura) no cálculo do gradiente pode ser visto na Figura 12a realizado sobre uma imagem de 18 *megapixels*. O número de pontos verdes (eixo vertical) representa



(a) (b)

Figura 12: Impacto das características imagem (a) e da resolução (b) no cálculo do gradiente.

o número de *pixels* cujo valor da intensidade é maior que o valor do gradiente, neste gráfico normalizado para valores entre zero e um.

Os resultados são bastante semelhantes excepto quando a imagem tem um nível de ruído considerável, pelo que se conclui que o ruído pode ter uma influência preponderante no cálculo do gradiente e desta forma deve ser estimado de forma a evitar cálculos errados.

5.6 Impacto da resolução da imagem no cálculo do gradiente

O impacto da resolução da imagem no cálculo do gradiente pode ser na Figura 12b. A resolução original da imagem em teste é de 18 *megapixels*, tendo sido reduzida várias vezes de forma a ser possível realizar este teste.

Analisando o gráfico pode-se concluir que em resoluções menores uma maior percentagem de pontos verdes é detetada, isto porque a operação de diminuição da resolução da imagem tende a fazer operações de *sharpening*.

6 Conclusões

Com a utilização da ferramenta de *software* proposta neste artigo espera-se que os resultados da automatização dos fluxos de trabalho no processo de seleção da melhor foto permitam ao utilizador ultrapassar de uma forma simples, precisa e eficaz os desafios apresentados, tornando-se assim a escolha da melhor foto numa experiência gratificante e motivadora.

Em relação às experiências realizadas pretende-se melhorar a parte da correlação de imagens visto que, tal como é apresentado na Figura 11b, este processo tem tempos excessivamente elevados quando em face de imagens de alta resolução. Na deteção de *focus* há também que ter em conta o nível de ruído existente de forma a melhorar a precisão deste procedimento.

O trabalho futuro passa pela implementação das restantes fases do *workflow* proposto seguido da construção da interface gráfica a ser apresentada ao utilizador. Como resultado final surgirá um produto de *software* que pode ser

standalone ou um *plugin* para um produto de gestão/manipulação de imagens como o Photoshop. Numa fase posterior pretende-se ainda aplicar técnicas de aprendizagem automática que permitam conhecer as preferências do utilizador de forma a simplificar ainda mais o processo de seleção da melhor fotografia.

Agradecimentos. Este trabalho foi parcialmente suportado pela Fundação para a Ciência e Tecnologia através do projeto estratégico PEst-UID/CEC/04516/2013.

Referências

1. H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.
2. W.-T. Chu, J.-C. Chen, and J.-L. Wu. Tiling slideshow: An audiovisual presentation method for consumer photos. *MultiMedia, IEEE*, 14(3):36–45, July 2007.
3. M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.
4. A. Foi, V. Katkovnik, and K. Egiazarian. Pointwise shape-adaptive DCT for high-quality deblocking of compressed color images. *European Signal Processing Conference*, 16(5):1–17, 2006.
5. Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting, 1995.
6. Itseez. OpenCV. http://opencv.org/, 2015. [Online; accessed 15-June-2015].
7. JEITA. *Exchangeable image file format for digital still cameras.* 2002.
8. J. Lee and K. Hoppel. Noise modeling and estimation of remotely-sensed images. In *Geoscience and Remote Sensing Symposium, 1989. IGARSS'89. 12th Canadian Symposium on Remote Sensing., 1989 International*, volume 2, pages 1005–1008, 1989.
9. S. Li and B. Yang. Multifocus image fusion using region segmentation and spatial frequency. *Image Vision Comput.*, 26(7):971–979, July 2008.
10. R. Liu, Z. Li, and J. Jia. Image partial blur detection and classification. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*, 2008.
11. D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.
12. Y. Lu, X. Feng, J. Zhang, R. Wang, K. Zheng, and J. Kong. A multi-focus image fusion based on wavelet and region detection. pages 294–298, 2007.
13. M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.
14. T.-A. Nguyen and M.-C. Hong. Filtering-based noise estimation for denoising the image degraded by gaussian noise. In *Proceedings of the 5th Pacific Rim Conference on Advances in Image and Video Technology - Volume Part II*, PSIVT'11, pages 157–167, Berlin, Heidelberg, 2012. Springer-Verlag.
15. E. Potapova, M. Egorova, and I. Safonov. Automatic photo selection for media and entertainment applications. *GRAPHICON-2009*, pages 117–124, 2009.
16. B. Su, S. Lu, and C. L. Tan. Blurred image region detection and classification. In *Proceedings of the 19th ACM International Conference on Multimedia*, MM '11, pages 1397–1400, New York, NY, USA, 2011. ACM.

O Conceito de Caderneta de Cromos e a Realidade Aumentada Aplicados aos Museus

Luís Costa[1✉], António Coelho[1, 2]

[1] FEUP - Faculdade de Engenharia da Universidade do Porto
mm12044@fe.up.pt

[2] INESC TEC – Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência
acoelho@fe.up.pt

Resumo. Os museus são uma peça central do património cultural, mas carecem de meios interativos que permitam acompanhar a recente evolução tecnológica, de forma a cativar um público mais jovem. Com enfoque em embarcações típicas portuguesas, um dos ex-libris da história portuguesa, e em particular em museus com uma vertente marítima, é objetivo do presente trabalho integrar o conceito de colecionismo e Realidade Aumentada (RA) numa solução de gamificação para museus. Previamente à definição da arquitetura da solução foi realizada uma pesquisa de forma a identificar os requisitos mais relevantes a introduzir. Neste sentido, é projetada uma caderneta de cromos digital assente em princípios inerentes à gamificação. A mecânica base é o colecionismo, tendo sido desenvolvida uma solução que possibilita adquirir um cromo quando se identifica um determinado elemento de um barco, diretamente através de uma funcionalidade de realidade aumentada. Assim, os cromos digitais obtêm-se ao ultrapassar desafios no contexto do museu, despoletando uma conjugação de elementos de realidade mista sob a forma de modelação tridimensional, texto e vídeos. Após a experiência concreta no museu, a caderneta de cromos digital permitirá desbloquear os modelos tridimensionais dos barcos cujos cromos foram completados, servindo assim ela própria como uma recompensa tangível, um *souvenir*. Na validação do protótipo desenvolvido, os resultados apontam para o manifesto interesse da solução.

Palavras-chave: colecionismo, gamificação, museus, realidade aumentada

1. Introdução

Os museus são um património singular que caracteriza a cultura de um país e de uma dada região, numa transversalidade entre a história, arte, ciência, transportes, o território e o mar, entre outros. Indiscutivelmente, visitar museus enriquece o nosso percurso cultural e, como resultado, temos vindo a assistir a um acréscimo gradual do número de visitantes segundo dados mais recentes do Instituto Nacional de Estatística (INE). No entanto, de acordo com a literatura e inquérito introduzido no âmbito do presente estudo, percepciona-se uma lacuna no que respeita aos meios interativos disponíveis para acompanhar a recente evolução tecnológica, de forma a efetivamente atrair um público mais jovem.

Partindo das embarcações típicas portuguesas, como um marco inalienável da história de Portugal, presentes em museus com uma vertente marítima, é objetivo do presente trabalho desenvolver uma solução que possibilite descobrir os barcos e as suas particularidades de forma mais aliciante. Para o efeito conjuga-se o conceito de colecionismo e realidade aumentada numa solução de gamificação para museus. A mecânica base é o colecionismo, tendo sido projetada uma solução que possibilita obter um

cromo quando se identifica um determinado elemento de um barco, diretamente através de uma funcionalidade de realidade aumentada. Assim, os cromos digitais resultam como recompensa de ultrapassar desafios no contexto do museu, despoletando uma junção de elementos de realidade mista sob a forma de modelação tridimensional, texto e vídeos. Após a experiência concreta no museu, a caderneta de cromos digital permitirá desbloquear os modelos tridimensionais dos barcos cujos cromos foram completados, tornando-se ela própria como uma recompensa tangível, um *souvenir*.

Após esta introdução, o artigo prossegue com a revisão de trabalho relacionado. A secção 3 releva os resultados do inquérito preliminar levado a cabo. Na secção seguinte descreve-se o desenvolvimento da solução, e a secção 5 compreende os resultados da validação do protótipo por especialistas. Por fim apresentam-se as conclusões e identificam-se linhas de trabalho futuro.

2. Trabalho relacionado

Para o presente estudo, o trabalho relacionado incide em aplicações tecnológicas disponíveis em museus, bem como nos conceitos de embarcações, colecionismo, gamificação e realidade aumentada.

Relativamente aos museus, tem-se vindo a assistir nos últimos anos à introdução de soluções tecnológicas sob diversos suportes, com vista a melhorar a experiência dos visitantes. Por um lado, os sistemas guiados têm vindo a incorporar elementos multimédia [1,2]. Por outro, incluem-se tecnologias de realidade mista para recriar peças tridimensionais [3,4,5], enfoque em pormenores do espólio museológico [6], [4,5] e recriação de exposições virtuais [2], [7], entre outros [8,9]. Em suma, a literatura confirma o potencial deste tipo de soluções [4], [10, 11]. No entanto, percepcionam-se como lacunas o facto da informação multimédia apenas estar disponível no momento da experiência, sem com isso incrementar a interatividade com o utilizador.

Nos últimos anos as embarcações típicas têm merecido um papel de destaque no património cultural português, onde esforços têm sido envidados para reconstituir o seu passado, desde as especificações técnicas à respetiva caracterização [12, 13]. Todavia, a atual forma de exposição não permite recriar uma experiência lúdica capaz de potenciar a sua aprendizagem.

Ao nível do conceito de colecionismo, inerente à caderneta de cromos, este pode ser definido como o processo de adquirir e possuir coisas de forma ativa, seletiva e apaixonada [14]. Diferentes públicos-alvo têm vindo a perpetuar o colecionismo, com os cromos de futebol e personagens do mundo do fantástico em tenra idade, ao universo do adulto com objetos colecionáveis e estratégias de marketing a eles direcionadas. Mas, perentoriamente o suporte físico tem-se sobreposto ao digital.

Em outra abordagem, a gamificação associa-se à utilização de elementos de jogo e técnicas de Game Design a contextos de não jogo, tendo recentemente emergido com uma base científica consolidada [15, 16, 17, 18, 19, 20], apesar de não existir uma definição universal [21, 22]. Atualmente áreas transversais têm acolhido os princípios de gamificação, desde o setor empresarial ao ambiente académico [16], [23], e resultados positivos têm vindo a ser comprovados [22], [15, 16], [18]. A implementação de uma estratégia de gamificação envolve um conjunto de elementos sob a forma de mecanismos e dinâmicas, onde os primeiros correspondem a regras e a dinâmica com o jogo/tarefa agradável, conforme quadro resumo com base na literatura [24], [17], [25]:

Tab. 1 - Quadro resumo dos elementos de jogo

Mecânicas	Dinâmicas	Motivos
Comportamento	Exploração	Curiosidade intelectual
Sistema de pontuações, troféus e medalhas	Coleção	Reconhecimento
Rankings	Competição	
Níveis		Status
Tarefas de grupo	Cooperação	
Pressão de tempo	Desafio	
Tarefas / quiz	Organização	Estimulação cognitiva
Avatars		Reconhecimento pessoal
Objetos virtuais		
Presentes virtuais		Recompensa

Ao nível das tecnologias de suporte, observa-se uma tendência crescente de combinar ambientes reais e virtuais, sob um espectro de realidade mista [26, 27]. Por realidade mista, entende-se um cenário híbrido onde o mundo real e os objetos virtuais são visualizados conjuntamente. Quanto à realidade virtual (RV), esta compreende um cenário onde o participante/observador está integralmente submergido num mundo artificial, com características reais clonáveis e obstáculos ultrapassados. Relativamente à realidade aumentada (RA) esta corresponde a uma sobreposição de objetos virtuais no mundo real, com potencialidades promissoras [28, 29, 30]. As respetivas aplicações têm vindo a ser acolhidas nas mais diversas áreas, tais como a educação, a cultura, a defesa nacional, a saúde, o entretenimento, o marketing, entre muitos outros domínios de atuação [17], [31, 32], [29], [33], [30], [34, 35, 36, 37, 38, 39].

3. Inquérito preliminar

Antes de definir a arquitetura da solução, o inquérito preliminar[1] visou aferir a adequação de introduzir uma aplicação interativa em museus, bem como avaliar a relevância de introduzir diferentes tecnologias e elementos. Adicionalmente, dado o lugar central atribuído à caderneta de cromos, foram questionados hábitos inerentes ao colecionismo. O público-alvo deste inquérito foram estudantes do mestrado multimédia da Faculdade de Engenharia da Universidade do Porto, bem como indivíduos que com eles se relacionassem e partilhassem o interesse no presente estudo. Os resultados obtidos encontram-se sumariados na figura 1, do ponto de vista da relevância.

[1] O inquérito preliminar pode ser consultado através do seguinte endereço: https://drive.google.com/file/d/0B9WuVC0P50w2NC1sN2M2OVItZ2M/view

Caracterização da amostra	
Dimensão	47 indivíduos
Idade	Média: 34 anos
Género	68% sexo masculino
Situação profissional	47% estudantes
Habilitações literárias	100% ensino superior (47% licenciatura)
Questões relacionadas com:	
Colecionismo	
Considera-se um colecionador?	Sim - 36% | Não - 64%
Atualmente, coleciona alguma coisa?	Sim - 40% | Não - 60%
Se não, qual o motivo?	Já colecionei, mas desisti entretanto - 57%
Se sim, o que coleciona?	Moedas - 21% | Selos - 16% | Cromos - 11%
Porque coleciona?	Gosto pessoal - 74%
Museus	
Costuma visitar museus?	Sim - 64% | Não - 36%
Se não, porquê?	Falta de interesse - 35% | Reduzida interatividade tecnológica - 29%
Se sim,	
Com que frequência?	3 vezes por ano - 47%
Que tipo de museus frequenta?	História - 36% | Marítimos - 17% (peso relativo)
Qual a importância de visitar museus?	Muito importante - 67%
De que forma visita os museus?	Em família - 60%
Tem por hábito adquirir souvenirs no museu?	Sim - 57%

Fig. 1 - Quadro síntese do inquérito preliminar

Assim, com 47 respostas válidas, as conclusões apontam para a importância de visitar museus, ocorrendo a visita no seio da dinâmica familiar, sendo que os que não visitam apontam como justificação a falta de interesse e a reduzida interatividade tecnológica. Uma parte substancial da amostra atualmente coleciona algo, ou pelo menos já o fez no passado, sobretudo motivados pelo gosto pessoal.

Numa etapa seguinte foi objeto de estudo a introdução de um caso genérico onde constava apenas a seguinte nota introdutória: “Imagine agora que, ao visitar um museu, tem acesso a descarregar uma aplicação gratuita no seu telemóvel. Com esta aplicação poderá no local obter informação adicional de determinadas peças icónicas, bem como levar para casa um conjunto de elementos virtuais para mostrar aos seus amigos e familiares.”. Em particular, foi pedido para avaliar a importância qualitativa da introdução de elementos na referida aplicação numa escala Likert de 1 a 5 [40], onde 1 corresponde a menos importante e 5 a mais importante. Os elementos referenciados para estudo são designadamente: realidade aumentada, virtual, animação tridimensional, texto, vídeos e caderneta de cromos (Fig. 2). Como se pode observar, a introdução de realidade aumentada surge como um dos elementos mais preponderante, seguindo-se com idêntico grau de importância a realidade virtual, animação tridimensional e vídeos.

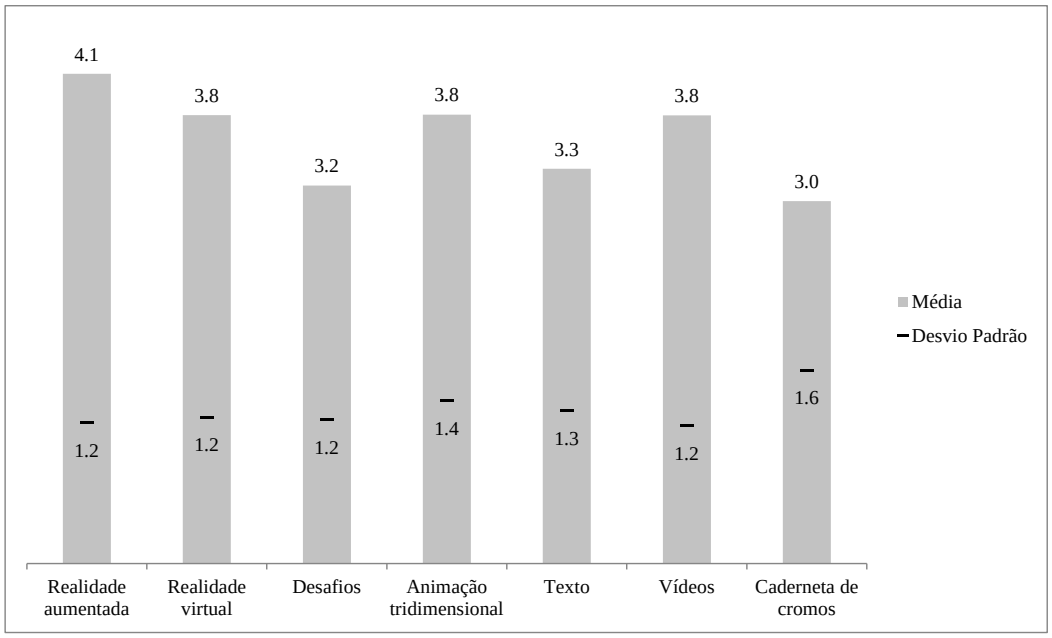


Fig. 2 - Análise gráfica do inquérito preliminar – caso de estudo

4. Desenvolvimento da solução

A solução proposta consiste numa caderneta de cromos digital, com aplicação ao contexto de um museu marítimo. A solução apresentada pretende ser implementada em dispositivos móveis. Como ponto de partida, foram objeto de estudo cinco embarcações típicas, onde a título exemplificativo se incluem: Ladra, Ílhava, Mercantel, Moliceiro e Barco do Mar. Este número de embarcações não é limitativo, mas entende-se como exequível para o desenrolar da experiência interativa num tempo aceitável.

Com base em elementos de gamificação será possível ultrapassar um conjunto de desafios. À medida que os desafios vão sendo superados torna-se possível desbloquear cromos digitais e ter acesso a um conjunto de informação multimédia ao nível de modelação tridimensional, vídeos e texto, em suporte de realidade aumentada e virtual.

O ponto de partida desenrola-se na recepção, onde o visitante é convidado a conhecer a solução, para decidir se entra no desafio. Dado o feedback obtido no inquérito preliminar, entende-se que, pelo facto das visitas ocorrerem sobretudo em família, esta possa suscitar maior interesse na conjugação de conhecimentos, ou competição. A figura 3 descreve as diversas etapas da solução e os elementos de gamificação que se encontram presentes, que serão descritos com maior detalhe seguidamente.

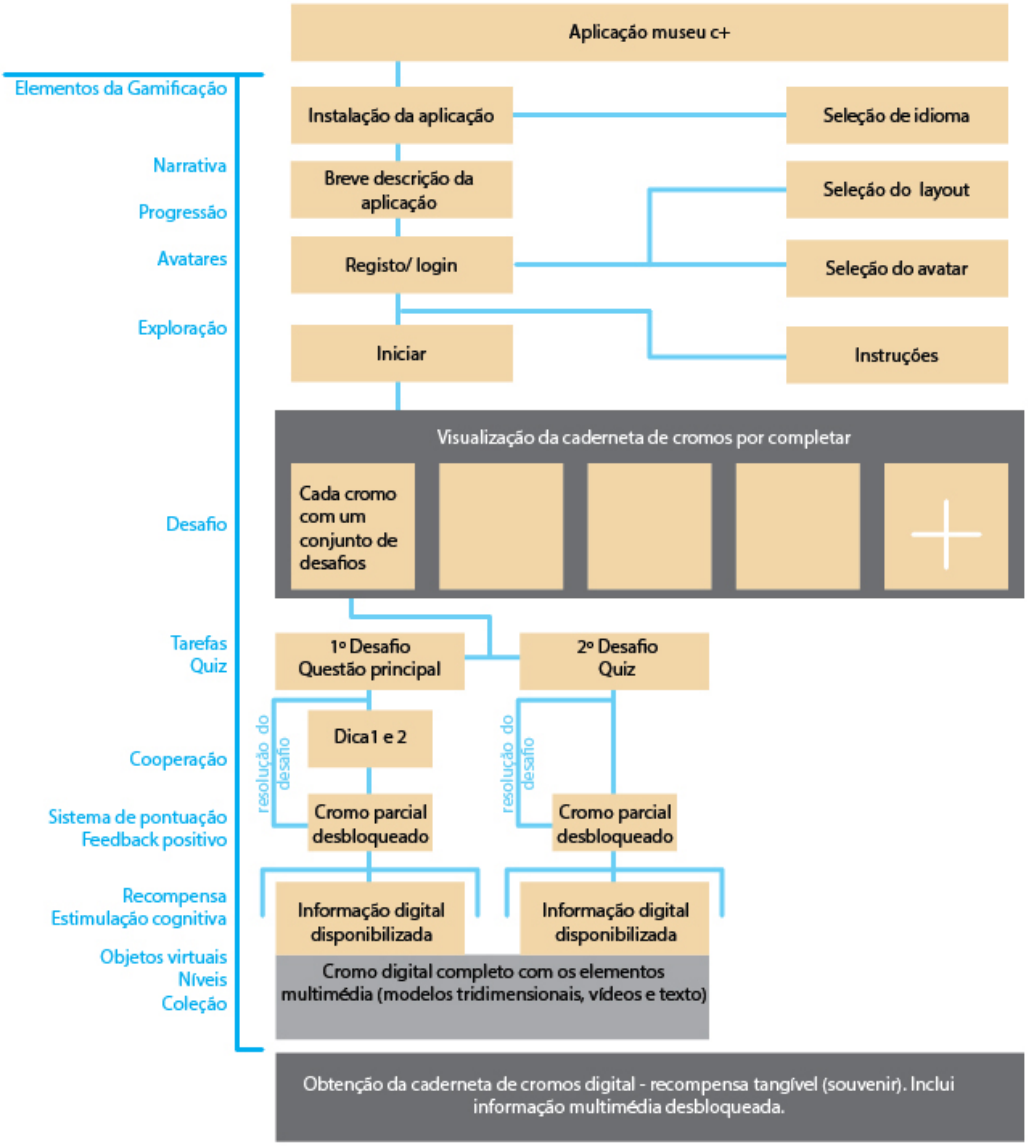


Fig. 3 Esquema da solução desenvolvida

Logo que seja feito *download* da aplicação gratuita, e após selecionar-se o idioma, surge uma breve introdução de qual é o objetivo da aplicação. Posteriormente surge o menu de registo / *login*, que poderá ser feito através do Google ou Facebook tornando assim possível publicitar a aplicação nestas redes sociais, numa perspetiva de transmedia. Após introdução dos dados, a fase seguinte consiste na escolha do tema com que o jogo surge desenhado: versão infantil ou histórica. No primeiro caso, as cores surgem mais vivas, a linguagem mais simples, e os conteúdos adaptados ao público infantil. Na componente histórica, o *layout* surge adaptado à cartografia histórica e com cores alusivas à época dos descobrimentos, em tons de castanho e com elementos fulcrais em tons de azul a simbolizar o mar e a ria. Numa fase seguinte surge a opção de selecionar o avatar para jogar, inspirado em trajes típicos. Numa etapa subsequente é mostrada a caderneta de cromos digital por preencher (Fig. 4), e uma breve nota de ajuda para explicar como proceder.

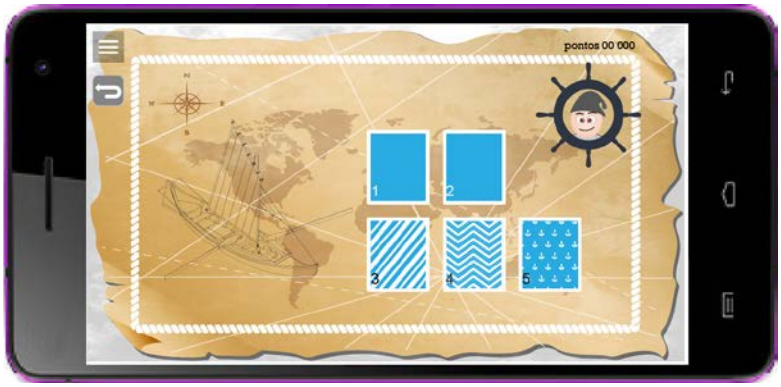


Fig. 4 *Layout* da caderneta de cromos por preencher

Para desbloquear a caderneta de cromos digital, é proposto um conjunto de desafios, que a seguir se apresentam de forma simplificada, e que poderá ser adequado a outros museus, futuramente.

A título exemplificativo encontra-se esquematizado na figura 5 o desafio proposto para a embarcação Ladra. A questão central consiste em "Encontre na Ladra os forcados da ré". Neste caso as ajudas disponíveis a despoletar correspondem a: (1) encontra-se na parte posterior da embarcação; (2) tem um determinado formato (desenho). O quiz que permitirá desbloquear a parte remanescente do cromo digital consiste em responder à seguinte escolha múltipla: "Quantas cavernas tem a Ladra?" A resposta correta neste quiz é o número 6. Neste caso a questão principal e o quiz estão intimamente relacionados no que respeita à estrutura vital que suporta a embarcação, e acessível à visualização direta por parte do visitante. A simplicidade dos pormenores técnicos desta embarcação, com um nome sui generis, torna os termos mais facilmente apreendidos.

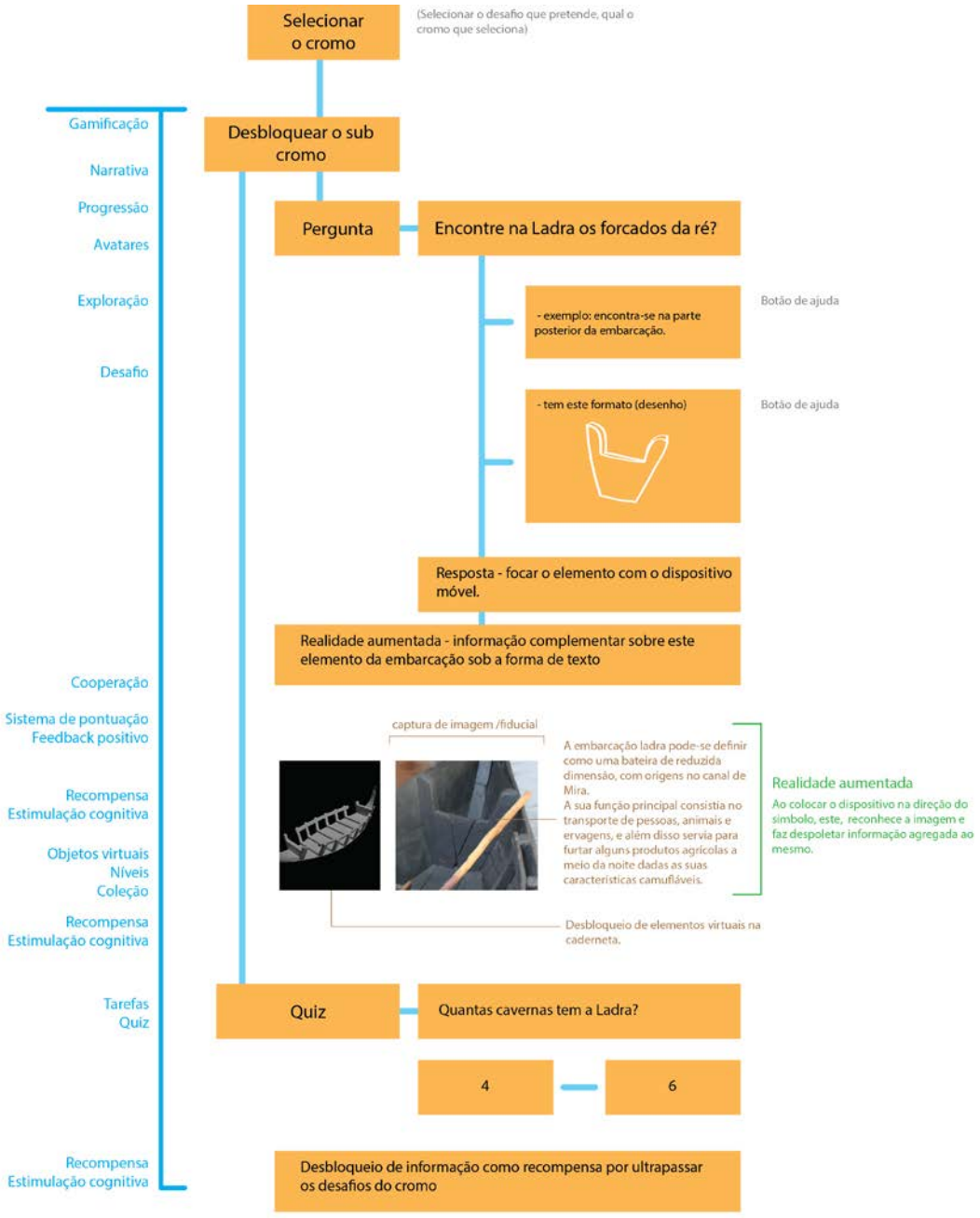


Fig. 5 Esquematização do desafio inerente à Ladra

Assim, começa por surgir uma questão principal relacionada com o primeiro desafio, apoiada por dois botões de ajuda, com pistas de teor diferente. Para estes desafios é necessário utilizar o modo de RA, no qual após apontar para a peça física correta da embarcação despoletará informação referente à utilidade da peça. Após ultrapassar a primeira parte do desafio, é desbloqueado o lado esquerdo do cromo. Neste caso, surge uma breve descrição do barco, e uma imagem elucidativa que ajudará a responder à segunda questão do desafio, e que permitirá desbloquear o lado direito do cromo digital. Nesta nova pergunta, o grau de dificuldade é substancialmente reduzido, consistindo numa escolha múltipla com duas opções. Após selecionar a opção correta, o cromo digital surge completo e novos conteúdos multimédia surgem disponíveis, tais como planos tridimensionais da embarcação (figuras 6 e 7) e uma pequena

animação de como se constrói a embarcação[2]. Assim, o utilizador passa de nível após ultrapassar cada desafio, ganhando acesso a um cromo digital (e conteúdo multimédia associado) como recompensa tornando mais cativante a exploração do museu e a apreensão real de conhecimento. Uma vez que se pretende promover a aprendizagem, os desafios conseguem ser sempre superados com as ajudas fornecidas. De notar que se encontra disponível a visualização dos conteúdos conquistados. Quase sem se aperceber o utilizador encontra-se a jogar num contexto real de museu, em paralelo com a aprendizagem de novos conhecimentos numa experiência interativa. E neste caso o participante / jogador sente que é a peça central do jogo. A progressão do jogo ocorre de forma intuitiva. A descoberta do museu para a coleção dos cromos torna a experiência mais interessante, ao mesmo tempo que se pretende promover a cooperação entre grupos de amigos ou família.

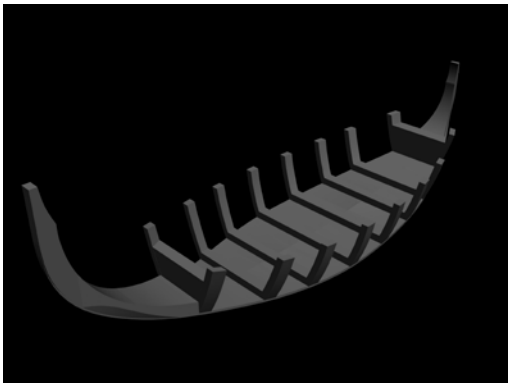


Fig. 6 Cavernas da Ladra em vista 3D

Fig. 7 Modelo tridimensional da Ladra

Ao longo dos cinco desafios, relativos a cinco embarcações típicas (Ladra, Ílhava, Mercantel, Moliceiro e Barco do Mar), diferentes níveis vão sendo ultrapassados e a pontuação é atribuída mediante o desbloqueio de cada cromo. No final da solução, os cromos digitais estão à disposição do utilizador, juntamente com a pontuação e um crachá de reconhecimento pelo esforço, em que lhe confere o grau de comandante da embarcação.

No global, diferentes elementos de gamificação vão estando presentes. No que respeita às mecânicas incluem-se um sistema de pontuação, crachás, níveis, tarefas, quizes, avatares e objetos virtuais. Relativamente às dinâmicas, estas residem na coleção, na exploração, na cooperação e no desafio. Ao nível dos motivos salienta-se a curiosidade intelectual, estimulação cognitiva, reconhecimento e recompensa.

5. Resultados

A solução desenvolvida foi validada por especialistas, designadamente o diretor de um museu marítimo e um estudioso na área das embarcações com livros publicados, através de um protótipo semi-funcional, por forma a identificar eventuais melhorias a acrescentar à aplicação final [41]. Com base na literatura anterior, no desenrolar da avaliação foi questionado um conjunto de métricas, designadamente:

[2] Uma versão preliminar da informação multimédia encontra-se disponível no seguinte endereço: http://lscostals.wix.com/luis-costa#!modelao-de-embarcaes/c1l47

atenção, relevância, satisfação, conformidade com a percepção do utilizador, usabilidade, efetividade, utilidade, facilidade de uso de acordo com instruções fornecidas, interesse, adequabilidade ao objetivo, etc.

Os resultados apontam para uma melhoria a introduzir ao nível das instruções. Globalmente é atribuída uma pontuação de excelente à relevância da solução apresentada e dos conteúdos, bem como um grau de satisfação elevado com a introdução da mesma no Museu. Neste sentido, antecipa-se o potencial da solução desenvolvida ao permitir introduzir interatividade no Museu e conduzir a um patamar acrescido de apreensão de conhecimentos.

Conclusões e trabalho futuro

Neste artigo apresenta-se o desenvolvimento de uma solução de caderneta de cromos digital aplicada ao contexto dos museus. Por ter sido identificada uma lacuna de interatividade tecnológica no património cultural por excelência, o próximo passo foi aliar princípios de gamificação sob o suporte da realidade aumentada a embarcações típicas. Para o efeito, o trabalho partiu de um inquérito preliminar para avaliar as características relevantes da solução a desenvolver. Após o desenvolvimento do protótipo inicial, os resultados foram avaliados em contexto prático por especialistas na matéria, e após essa análise melhorias serão introduzidas e a aplicação será desenvolvida. De forma inequívoca os resultados comprovam o interesse na solução, numa perspetiva de introdução de interatividade tecnológica nos museus e fomento da aprendizagem num enquadramento de embarcações típicas.

Relativamente ao objetivo do presente trabalho, observa-se que a combinação de realidade aumentada e virtual permite apresentar de forma interativa e inovadora novos conteúdos multimédia no espaço museológico, onde as embarcações tradicionais se inserem. Em paralelo, a abordagem de gamificação pela qual se norteia a solução induz uma aprendizagem mais efetiva, onde à medida que novos desafios vão sendo ultrapassados novo conhecimento é adquirido, culminando com a recompensa última de uma caderneta de cromos digital que se associa a um comportamento de colecionismo.

As principais contribuições deste trabalho unificam uma perspetiva académica e prática. Por um lado, propõe-se uma nova abordagem inovadora baseada na interligação da gamificação com tecnologias de realidade aumentada e virtual e com o suporte de conteúdos multimédia. Para o efeito, os princípios do colecionismo assentam numa caderneta de cromos digital. Por outro, a vertente prática induz a introdução de uma nova solução tecnológica para a visita a museus, conciliando a inovação de explorar interativamente embarcações típicas portuguesas, mas que poderá ser alargada a outro tipo de museus. Adicionalmente, foram desenvolvidos os desenhos e modelos tridimensionais das embarcações que estão publicados num livro [12].

Encontra-se em desenvolvimento a aplicação móvel, em Unity 3D, incorporado o *software* Qualcomm Vuforia. Esta biblioteca de RA, para além de permitir desenvolver aplicações de RA em dispositivos móveis com sistema Android e iOS, tem como vantagem adicional o uso de marcadores mais complexos, que se tornam menos sensíveis ao movimento e as imagens ficam mais estáveis [2]. Numa perspetiva de trabalho futuro, a definição de uma caderneta digital traz uma importância acrescida quando pensamos não só na museologia marítima mas sim na inclusão de outro espólio museológico. Pretende-se

desta forma possibilitar as bases para a criação de uma caderneta de cromos digital aplicada aos museus de Portugal, incentivando o utilizador a descobrir todo o património português. A interligação do colecionismo com a gamificação e as novas tecnologias potenciam uma perspetiva nova de visitar museus, onde recompensas digitais podem ser levadas como souvenir. Posteriormente, pretende-se aplicar uma abordagem transmedia, incorporando a narrativa museológica através de outros media, tais como conteúdos multimédia e bibliografia em *web sites* de referência e nas redes sociais. A aplicação poderá também servir de canal para divulgação de notícias sobre novas exposições e eventos a ocorrer. Finalmente, esta aplicação poderá estar ligada a um livro digital pronto a ser impresso e compilar diferentes momentos da experiência museológica do utilizador.

Bibliografia

1. Chen, C.Y., Chang, B.R., Huang, P.S.: Multimedia augmented reality information system for museum guidance. Personal and Ubiquitous Computing, 18, 315-322 (2014).
2. Ramirez, M., Ramos, E., Cruz, O., Hernandez, J., Perez-Cordoba, E., Garcia, M.: Design of interactive museographic exhibits using Augmented reality. 23rd International Conference on Electronics, Communications and Computing, CONIELECOMP, Cholula, Puebla. 1-6 (2013).
3. Jiménez Fernández-Palacios B., Nex, F., Rizzi, A., Remondino, F.: ARCube—The Augmented Reality Cube for Archaeology. Archaeometry, 57, 250-262 (2015).
4. Choi, H.S.: The conjugation method of augmented reality in museum exhibition. International Journal of Smart Home, 8, 217-228 (2014).
5. Keil, J., Pujol, L., Roussou, M., Engelke, T., Schmitt, M., Bockholt, U., Eleftheratou, S.: A digital look at physical museum exhibits: Designing personalized stories with handheld Augmented Reality in museums. First International Congress on Digital Heritage, DigitalHeritage, Marseille, 685-688 (2013).
6. Clini, P., Frontoni, E., Quattrini, R., Pierdicca, R.: Augmented reality experience: From high-resolution acquisition to real time augmented contents. Advances in Multimedia, vol. 2014, Article ID 597476 (2014).
7. Jevremovic, V., Petrovski. S.: MUZZEUM - Augmented Reality and QR codes enabled mobile platform with digital library, used to Guerrilla open the National Museum of Serbia. 18th International Conference on Virtual Systems and Multimedia: Virtual Systems in the Information Society, VSMM, Milan, 561-564 (2012).
8. Carmo, M.B., Cláudio, A.P.: 3D virtual exhibitions. DESIDOC Journal of Library and Information Technology, 33, 222-235 (2013).
9. Chen, S., Pan, Z., Zhang, M., Shen. H.: A case study of user immersion-based systematic design for serious heritage games. Multimedia Tools and Applications, 62, 633-658. (2013).
10. Yoon, S. A., Wang, J.: Making the Invisible Visible in Science Museums Through Augmented Reality Devices. TechTrends, 58(1), 49-55 (2014).

11. Takahashi, T.B., Takahashi, S., Kusunoki, F., Terano, T., Inagaki, S.: Making a hands-on display with augmented reality work at a science museum. 9th International Conference on Signal-Image Technology and Internet-Based Systems, SITIS, Kyoto, 385-390 (2013).
12. Fonseca, S.: Embarcações lagunares: Bateiras e Artes – Arquitetura naval lagunar. (Em impressão) (2015).
13. Fonseca, S.: Embarcações que tiveram berço na laguna – Arquitetura naval lagunar. Porto, Papiro Editora. ISBN 978-989-636-587-5. DL 331260/11 (2011).
14. Belk, R.: Collecting as Luxury Consumption: Effects of Individuals and Households. Journal of Economic Psychology, 16(3), 477- 492 (1995).
15. Armstrong, D.: The new engagement game: the role of gamification in scholarly publishing. Learned Publishing, 26, 253-256 (2013).
16. Jagoda, P.: Gamification and Other Forms of Play. Boundary 2-an International Journal of Literature and Culture, 40, 113-144 (2013).
17. Simões, J. Redondo, R.D., Vilas, A.F.: A social gamification framework for a K-6 learning platform. Computers in Human Behavior, 29, 345-353 (2013).
18. Kim, J.T., Lee, W.H.: Dynamical Model for Gamification: Optimization of Four Primary Factors of Learning Games for Educational Effectiveness. Computer Applications for Graphics, Grid Computing, and Industrial Environment, 351, 24-32 (2012).
19. Matsumoto, T.: Possibility of E-Learning Education That Uses the Gamification. 5th International Conference of Education, Research and Innovation, 3310-3314 (2012).
20. Deterding, S., Dixon, D., Khaled, R., Nacke, L.: From game design elements to gamefulness: defining "gamification". Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments. Tampere, Finland: ACM (2011).
21. Ferrara, J.: Games for Persuasion: Argumentation, Procedurality, and the Lie of Gamification. Games and Culture, 8, 289-304 (2013).
22. Seaborn, K., Fels, D. I.: Gamification in theory and action: A survey. International Journal of Human Computer Studies, 74, 14-31 (2014).
23. Zichermann, G. Linder, J.: Game-based Marketing: Inspire Customer Loyalty through Rewards, Challenges, and Contests. Wiley, Hoboken, NJ. ISBN: 978-0-470-56223-9 (2010).
24. Blohm, I., Leimeister, J.M.: Gamification. Design of IT-Based Enhancing Services for Motivational Support and Behavioral Change, Business & Information Systems Engineering, 5(4), 275-278 (2013).
25. Neeli, B.K.: A Method to Engage Employees using Gamification in BPO Industry. Third International Conference on Services in Emerging Markets, 142-146 (2012).
26. Billinghurst, M., Clark, A., Lee, G.: A survey of augmented reality. Foundations and Trends in Human-Computer Interaction, 8, 73-272 (2014).
27. Milgram, P., Kishino, F.: A Taxonomy of Mixed Reality Visual Displays, SPIE Telemanipulator and Telepresence Technologies, Vol. 2351 (1994).

28. Linaza, M., Gutierrez, A., García A.: Pervasive Augmented Reality Games to Experience Tourism Destinations, Information and Communication Technologies in Tourism 2014, Z. Xiang & I. Tussyadiah, Eds. Springer International Publishing, 497-509. (2013).
29. Chi, H., Kang, S., Wang, X.: Research trends and opportunities of augmented reality applications in architecture, engineering, and construction. Automation in Construction 33, 116-122 (2013).
30. Rambli, D., Matcha, W., Sulaiman, S. Nayan, M.: Design and development of an interactive augmented reality edutainment storybook for preschool. IERI Procedia 2, 802-807 (2012).
31. Rose, K.J., Konig, M., Wiesbauer, F.: Evaluating success for behavioral change in diabetes via mHealth and gamification: MySugr's keys to retention and patient engagement. Diabetes Technology & Therapeutics, 15, A114. (2013).
32. Goehle, G.: Gamification and Web-based Homework. PRIMUS, 23(3), 234-246 (2013).
33. Chantzi, A.E., Plessa, C., Gkanas, I.C.: Design and Development of Educational Platform in Augmented Reality Environment using Gamification to enhance Traditional, Electronic and Lifelong Learning Experience. Local Proceedings of the Sixth Balkan Conference in Informatics Thessaloniki, Greece (BCI), 92 (2013).
34. Nee, A., Ong, S., Chryssolouris, G., Mourtzis, D.: Augmented reality applications in design and manufacturing. CIRP Annals - Manufacturing Technology, 61, 657-679 (2012).
35. Li, W., Grossman, T., Fitzmaurice, G.: GamiCAD: a gamified tutorial system for first time AutoCAD users. In: Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology. Presented at UIST'12. ACM, Cambridge, MA, 103-112 (2012).
36. Cafazzo, J.A., Casselman, M., Hamming, N., Katzman, D.K., Palmert, M.R.: Design of a mHealth app for the self-management of adolescent type 1 diabetes: a pilot study. Journal of Medical Internet Research. 14(3) e70 (2012).
37. Jaramillo, G., Quiroz, J., Cartagena, C., Vivares, C., Branch, J.: Mobile Augmented Reality Applications in Daily Environments. Revista EIA (14), 125 (2010).
38. Lam, A., Chow, K. H., Yau, E., Lyu, M.: ART - Augmented Reality Table for Interactive Trading Card Game. In Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications, 357-360 (2006).
39. Billinghurst, M., Kato, H., Poupyrev, I.: The MagicBook: a transitional AR interface. Computers & Graphics, 25, 745-753 (2001).
40. Likert, R.: A Technique for the Measurement of Attitudes, Archives of Psychology, 140, 1-55 (1932).
41. Santos, M. E. C., Chen, A., Taketomi, T., Yamamoto, G., Miyazaki, J., Kato, H.: Augmented reality learning experiences: Survey of prototype design and evaluation. Ieee Transactions on Learning Technologies, 7(1), 38-56 (2014).

Physical Intrusion Detection and Prevention for Android Smartphones

Joana Velho, Diogo Marques, Tiago Guerreiro, and Luís Carriço

Faculty of Sciences of the University of Lisbon
jvelho@lasige.di.fc.ul.pt, {dmarques, tjvg, lmc}@di.fc.ul.pt

Abstract. Authentication mechanisms are useful when a device is lost or stolen, but ineffective when it comes to preventing friends and family from snooping through contents. Most unlock authentication methods are vulnerable to observation attacks than can easily be performed by those in a close social circle. Moreover, unlock authentication does not address the common use case of device sharing. Intrusion Detection and Prevention Systems (IDPS) are based on the assumption that a system will eventually be attacked, and are widely used in network systems as an additional security measure that works around authentication flaws. In this paper, we present and evaluate the adequacy of an inconspicuous IDPS for Android smartphones, intended to dissuade socially-close adversaries from snooping through device contents. This system runs on the background and attempts to determine, through face recognition, if the device is being operated by the owner. If it is not, it starts recording user actions, which can later be reviewed by the owner. We conducted a laboratory study (n=12) to examine users concerns over other people looking through their data, and to present the system to participants. We also conducted a field study (n=10), where participants used the system for an extended period of time, in order to understand how they adopted it. Results indicate that the IDPS approach addresses previously unmet needs, namely by offering a security measure that does not require users to expend effort in every interaction with the device.

Keywords: IDPS, Inconspicuous, Mobile Devices, Adoption

1 Introduction

Mobile devices, such as smartphones and tablets, have become ubiquitous, and keep data on many aspects of users' lives. For that reason, many users set up unlock authentication to inhibit others from accessing device contents. Modern mobile devices offer a variety of authentication mechanisms, including some based on secrets, such as PIN, password and pattern, and some based on biometrics, such as face recognition. Unlock authentication is useful when a device is lost or stolen. However, recent studies suggest that unlock authentication is ineffective in protecting mobile contents from close adversaries [13]. On one hand, the most popular unlock authentication methods that are based on secrets are vulnerable to simple physical attacks, that can easily be performed by those in a

close social circle. On the other hand, some users quit or never configure unlock authentication because they consider inconvenient to enter a code every time they use their devices. Furthermore, unlock authentication does not address the common use case of device sharing.

In order to prevent family and friends from snooping through device contents, and to address the device sharing case, we designed and developed and intrusion detection and prevention system (IDPS) for Android smartphones. This software recognizes the owner through face recognition, and starts recording user actions when someone else uses the device. Later, the device owner can review the recordings. We designed our proposed IDPS to be inconspicuous, by running on the background and not disrupting the normal use of the device.

In this paper, we present the concept and architecture of our proposed system, as well as implementation details. We then present the results of a study consisting of semi-structured interviews with twelve smartphone users. The study was conducted in order to explore motivations for device sharing, and to identify defenses and emerging concerns. In this study, we also presented a prototype to participants, in order gather their perceptions on how such a system could be put to use. Finally, we present the results of a field study that was conducted to better understand how participants really adopt this system in their social context. Ten participants were recruited for this study, and used an instrumented version of our software for nine days. We assessed the usefulness of this approach and whether it addresses user requirements. We also used this study to find remaining usability problems. Overall, the IDPS approach was found to be useful, and to cater to user's desire to have security without having to incur in constant effort and vigilance.

2 Related Work

Smartphones and tablets are used for a many purposes, including to play games, access social networks, make phone calls, send text messages and shop online. With the emerging *Bring Your Own Device* trend, where employees are encouraged to use their devices to access enterprise data and systems, mobile devices also now commonly store sensitive work products [10].

Users are aware of the sensitivity of the data stored on their devices, and are concerned about security threats [5]. Recent studies indicate that unauthorized access by socially-close adversaries is not only not an uncommon occurrence, but may have a particular negative impact on users. In a survey of internet users, 14% of participants reported that their mobile devices were used by someone else without permission, and 9% admitted that they have used someone else's device without permission [13]. In another recent study, 70% of participants indicated a preference for preventing socially-close individuals from accessing some functionality on their phone [9], when confronted with a tool that provided that ability. Among the reasons invoked to use unlock authentication, users often cite physical threats, like the risk of loss or theft, or the desire to avoid family and

friends from snooping through contents or past unauthorized access experiences [7].

Unlock authentication mechanisms are intended to protect mobile devices in the event of theft or loss, but they are not appropriate to avoid friends and family from snooping through contents. Most unlock authentication methods are vulnerable to observation attacks than can easily be performed by those in a close social circle. The most popular unlock authentication mechanisms, which are based on secret codes, are susceptible to shoulder-surfing attacks, where someone could find out the access code just by looking when it is being entered [15]. Unlock authentication mechanisms, specially patterns, are also vulnerable to smudge attacks [2]. Interactions with the touch screen leave oily residues from the fingers; an attacker can observe the marks and often infer the secret code. Such observation attacks can easily be performed by those in a close social circle, for instance friends and family. Furthermore, many users choose not to lock their devices or give up on unlock authentication. In a recent large-scale study of smartphone users, 42% indicated that they didn't lock their devices. The most cited reason was that locking was too much of a hassle. [7]

Mobile devices users often share their devices with others for specific tasks, such as making phone calls, sending text messages and playing games [12]. As a result, sometimes the owners cannot control what others are doing on their devices, even if momentarily. Unlock authentication does not address this common use case. It will not avoid others from snooping through contents, since it is an all-or-nothing access control mechanism. Protecting contents from those in a close social circle should start with and understanding of how people commonly use their devices, and what actually worries them.

A proposal similar to ours is *continuous authentication*, in which the operator's identity is continuously monitored during the interaction with the device. Crawford *et al.* proposed a continuous and transparent authentication framework for mobile devices based on keystroke dynamics and speaker verification [6]. This framework associates identity confidence levels to tasks. The confidence value is re-calculated from biometric data acquired while devices are being used. Itus [11], is a related approach, and provides implicit authentication by continuously authenticating the device owner based on biometric behavior using accelerometer, touch and keystroke dynamics. The intrusion detection aspect of our approach is similar to these proposals, in the sense that it also captures biometric data to continually identify the operator.

FaceProfiles is another concept similar to our proposal, which associates different access permissions to groups of contacts, and when a new user is detected, through face recognition, permissions are recalculated and the user interface adapts accordingly by showing only applications allowed [8]. Our approach is similar, in the sense that it is also uses facial recognition to identify users, and that it targets sharing among socially-close adversaries, but differs in that the reaction is not multi-user support, but logging.

3 A user-facing IDPS

3.1 Concept

Intrusion detection and prevention systems are based on the assumption that a system *will* be attacked. Their main purpose is to detect intrusions and prevent damages. The intrusion detection model may be suitable for mobile devices as a defense against physical attacks performed by socially-close adversaries.

Drawing on this model, we designed and developed an inconspicuous physical IDPS for Android smartphones. It can prevent, detect and react to intrusions. It is capable of identifying if the device is being operated by the owner, and if not, reacts by recording user actions. The recordings are made available for later review. It offers the device owner the opportunity to know who used his device and for what purpose.

Intrusion detection. We propose detecting if the device is being used by its owner or by someone else, by continuously authenticating the operator. Continuous authentication using biometric characteristics does not require the user to perform any specific action. We specifically propose using facial recognition to identify the device's owner. While the user is interacting with the device, the system is, periodically and inconspicuously, taking pictures using the front-facing camera. This way the user can operate the device normally while intrusion detection runs on the background.

Intrusion reaction. Intrusion reaction is a set of actions performed by the system when it detects an intrusion [14]. This system's reaction to intrusions is to record users actions, in such a way that those recordings can be later audited by the owner.

Intrusion prevention. Surveillance cameras protect people, places and objects by constantly monitoring physical spaces. It is well know that just the awareness of their existence inhibits misbehavior. Mirroring this concept, we propose making the device display a permanent warning (as a notification), informing the operator that pictures will be taken and actions on the device will be recorded. In a close social context, even if the attacker hides his/her identity to the camera, it is likely that the owner is able to perform the identification given other context (e.g., time and even the details of the attack). Our system does not place any barriers that prevent an adversary from accessing the device. Yet it supports a password lock to access this mobile application.

3.2 Implementation

Our system is composed by a service module, an intrusion detection module, a recording module, a data repository and a user interface (Fig. 1).

When our application is running, the service module is responsible for coordinating the intrusion detection and recording modules. When the device is unlocked after a period of inactivity, the service module starts the intrusion detection module and stops it when the device is again locked. The intrusion

detection module is responsible for taking pictures inconspicuously and periodically, for processing face recognition analysis, and for then reporting the results to the service module. The service module stores the captured pictures and reacts depending on the face recognition result, by instructing the recording module to start or stop. The recording module is responsible for capturing users actions on the device.

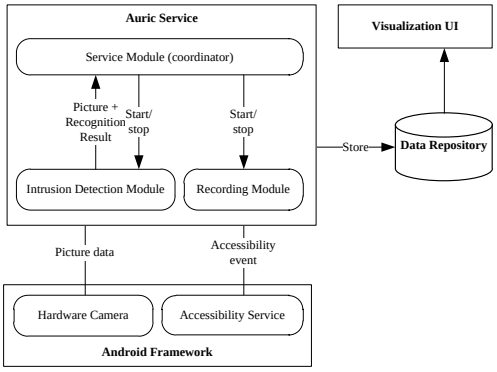


Fig. 1. Overview of the system's Android implementation.

3.3 Service Module

This module is responsible to start and stop recordings and for the intrusion detector. It has a receiver which listens to three types of Android system events and is notified when those events occur. It listens to "screen on" events, that are sent when the device wakes up and becomes interactive; "user present" events, that are sent when the user is present after the device wakes up, for instance after unlock; and "screen off" events, that are sent when the device becomes non-interactive. When a user starts to interact with the device, the background service is notified and starts the intrusion detector and waits for the face recognition result. If the device owner was not recognized, the background service will prompt the recording module to activate. If the owner is recognized by the intrusion detector, the background service will prompt the recording task to stop.

The service module acts as a coordinator of intrusion detection and user action recording. This way, the intrusion detection and recording modules are independent from each other. This module is also responsible for launching a notification informing the user that a intrusion was detected.

3.4 Intrusion Detection Module

This module is responsible for capturing pictures, processing face recognition analysis, and deciding the device is under a possible intrusion or not. Specifically, this module includes a task that awakes periodically and takes pictures

inconspicuously, using the front-facing camera. After that, the task performs face preprocessing, face detection and then face recognition. If the face recognition analysis indicates that the picture taken matches the owner's face with a configurable level of confidence, a possible intrusion is considered to be underway. Finally this module reports the result to the coordinator, which is the background service task.

We implement face recognition procedures, for instance face preprocessing, detection and recognition, using the OpenCV Library[3] and JavaCV[1]. OpenCV is an open source computer vision and machine learning software library that offers optimized algorithms to detect and recognize faces and also offers a SDK for Android. JavaCV is a Java interface to OpenCV which allow us to implement some features in Java instead of C++.

3.5 Recording Module

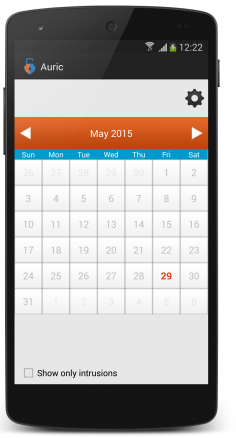
This module is responsible of recording users actions. It gathers data and processes it to a representation that is suitable for auditing. It supports two different methods of recording users interactions: *screen recording* and *event-based recording*.

The screen recording method produces a video of user activity on the device. Each frame of the video is like a screenshot taken while using the device.

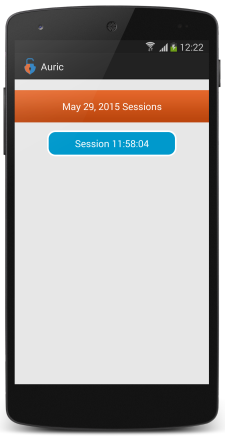
The event-based recording method produces a list of applications accessed and details about users interaction in each application. This method relies on events provided by Android's accessibility API. Accessibility events are messages about user interactions with visual interface components in an application. Those events help to produce a detailed log of users interactions with the device. This module communicates with an accessibility service that listens to specific accessibility events, such as text changing, view clicked, view selected, view scrolled and others. When one of those events occur, the accessibility service is notified and sends that information to this module to be processed and stored.

3.6 Visualization

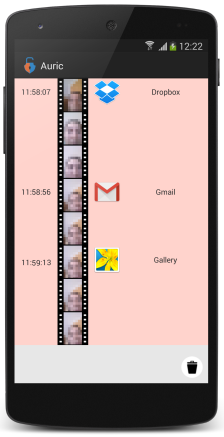
We designed an interface that allows the owner to easily access the recordings of intrusions detected (Fig. 2). We designed a calendar view showing the current month with the current day in bold; days where there were suspected intrusions are selectable and marked in red (Fig. 2(a)). By clicking on one of those days, a list of intrusion sessions will appear (Fig. 2(b)). Upon selecting a session, the recording of intruder's actions is shown. The way in which sessions are shown depends on recording method selected. If the screen recording method was used, a video capture of what happened on the screen is played, with the pictures taken with the front-facing camera rolling in the upper left corner. If instead the recording was event-based, a time-line of apps that were opened is shown, alongside a camera roll of pictures taken with the front-facing camera (Fig. 2(c)). Upon selecting an item on the list, all actions that were performed while using that app are shown (Fig. 2(d)).



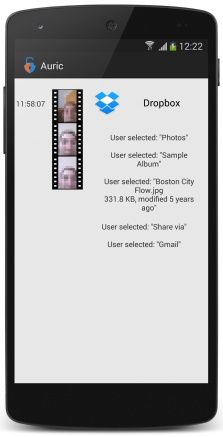
(a) Calendar View: the days when suspected intrusions occurred are marked in red.



(b) List of sessions from a day



(c) Time-line of apps used in a session, decorated with photos taken with the front-facing camera



(d) Details View: shows user actions within an app

Fig. 2. Visualization of logged activity on an Android smartphone.

4 Laboratory Study

We conducted a laboratory study in order to (1) learn about precautions that users have with mobile devices, to (2) explore concerns they have about people looking through their data, (3) to explore motivations for device sharing, and finally to (4) present the system to participants and understand how they would adopt it.

In this study, as well as in the next, we opted for a version of the system that only uses event-based logs. The screen recording method required devices to be rooted, and hence, we reasoned, would not realistically be adopted by many users.

4.1 Procedure

We conducted in-depth semi-structured interviews with twelve participants that use mobile devices, such as smartphones and tablets, on a daily basis. The study started with a set of questions about precautions that users have with mobile devices and their concerns over physical attacks. We then introduced our system's purpose and features. Next, we prompted participants to enroll in the system, and conducted a dramatization of two possible situations where our system can be useful, as follows:

Use case 1

1. The device is left unattended in a room for a few moments.
2. An attacker unlocks it and checks e-mail and text messages.
3. Before the device owner returns, the attacker leaves it where it was.
4. The device owner checks if someone used his device in his absence.

Use case 2

1. The owner searches for a photo in the device.
2. The owner hands the phone to another person to show that picture.
3. When unsupervised, that person sends it via e-mail.
4. The device is returned to its owner.
5. The device owner checks if the device was only used to view that photo.

After the dramatization, where participants acted as the owner, we conducted an exit interview, where we started by asking participants for general comments, and then probed specifically about our system's usefulness, and how they would adopt it, if at all.

4.2 Analysis

We recorded audio of the interviews and then a researcher transcribed it for analysis. The analysis of the interviews was done using thematic coding inductively. [4]. Two researchers used the first four interview transcripts for code discovery and then independently developed code books. Then, they met and agreed on a preliminary set of codes. Afterwards, the same two researchers re-coded four

interviews and compared the results, and by consensus, agreed on an extended set of codes. The two researchers then coded all the remaining interviews. Reliability was measured in the end and found to be acceptable (Cohen's $\kappa = .92$). The reported results are the marginal frequencies found by one of the coders.

4.3 Results

Current Usage All recruited participants, except one, considered their devices to be private, because they contain private or personal data.

A quarter of participants indicated that someone used their devices without authorization or to access unauthorized content.

> "There was a time that I lent the phone for someone else to play, and I ended up discovering that he was not playing. Suddenly I peeked and I saw that person was reading my text messages." (P1)

Device Sharing All participants reported that they have shared their devices with someone else to perform specific tasks: to show something (10/12), to make phone calls (5/12), to send text messages (4/12), to play games (2/12), to surf the internet (2/12), and for other purposes (3/12), such as using camera or operating the music player.

We identified some defenses used by the participants, including not handing the device to share contents, instead just showing it on their hand (5/12); keeping the mobile device in close proximity at all times (5/12); taking precautions with the accounts on the device, for instance, logging out (4/12), keeping close supervision when someone else is using the device (4/12) and sharing the device with the target application already open (1/12). The large majority of participants (10/12) commented on how trust affects attitudes towards the use by others.

> "I only give the device to someone I trust." (P8)

Adoption We tried to understand how participants planned to use the technology, if it was available to them. Responses indicate it often depends on the specific situation and on the nature of relationships with others. Half the participants intended to adopt the technology for deterrence. They foresaw using the application in such a way as to inhibit others to misbehave, for instance by informing them that the application was installed or by having it show notifications.

> "To be a deterrent method. Don't touch or I will know." (P12)

Almost all participants (11/12) intended to use our system for passive discovery of intrusions. They suggested using the application to discover misbehavior by others, but without any explicit intention to change their own behavior in order to catch intruders. A significant portion (4/12), however, indicated that they would use our system for entrapment, intending to actively create situations where others might be caught misbehaving.

"Maybe I would leave the phone on the table on purpose." (P5)

Only one participant intended to inform everybody that our system was installed. Three indicated that they wouldn't inform anyone. The majority (8/12), however, said they would inform some people but not others. Similarly, when asked if they would set up notifications, only one participant wanted to always show them. The majority (8/12) didn't want to show notifications at all, and 3/12 wanted to show them depending on the situation. Two participants indicated that they wouldn't tell anyone or show notifications because it would be embarrassing if others knew they were being recorded. Four participants commented on how anonymity affects behavior, indicating that if they were to let others know, they might act differently.

> "They do not know what they are doing is being monitored and it is more likely that people will do something, that maybe would not with the user's supervision." (P3)

A quarter of participants expected to catch people using their devices without permission if they were using our application; half did not, and the remaining were unsure.

Suggestions Prompted to give suggestions of ways to react to another person using the device, 3/12 participants suggested locking the device, 2/12 suggested to restrict access to certain contents, 2/12 to lock only if there was a high risk (e.g. when the other user does something that is considered sensitive). Some participants (3/12) also suggested that a notification to an external service could be sent, for instance via e-mail.

Advantages We asked participants if they saw any advantages of this approach in comparison to the security they already have in place. Two participants indicated that our application would be useful to control damages, by informing what contents were accessed.

> "To anticipate damage, if it is something secret, such as documents. People can have a contingency if they know what happened." (P3)

Half the participants indicated that our system could be used as an additional security measure; for instance, that it could be used along with unlock authentication.

The vast majority of participants, however, saw as the main advantage the ability to regulate social relations, for instance, using our system to "know who your friends are".

> "If I had this application, it would be easy to see who to trust and who could not be trusted" (P10)

> "I think we should just live surrounded by the people that we trust and they would not do this to us [snoop]. This app would help me identifying those people and have control over who you consider as a friend." (P5)

We also asked participants if they would like to leave with our application installed on their devices. Most did (10/12), but still some didn't (2/12). In those cases, technical difficulties were cited, e.g. not having a front-facing camera.

From this study, we conclude that users are interested in the possibility of auditing unauthorized access to contents. Users, the study suggests, could also use this technology to deter people in close social circles from even considering the possibility of snooping through device contents. Most of participants would adopt this technology as a way to regulate social relations, figuring out "who your friends are". In fact, it seems that many participants were more concerned with that than in keeping privacy.

5 Field Study

The main goals of field study were to perceive how participants would actually adopt this system, assess whether the concept is useful and whether it is appropriate to user requirements, and also to detect usability problems. For this study, we installed an instrumented version of our application on the participants own devices. The study lasted nine days, at end of which the application was removed. Data was gathered from three meetings with participants.

5.1 Apparatus

The participants used an alternative version of our system that did not take pictures; hence, it didn't detect intrusions, and simply recorded all interactions. In the field study version, participants also couldn't review logs on their own, only during the meetings with the researchers, which had a master password. We chose to prepare this special version because it would be unethical to record data from people that did not agree to participate, which would be the case of possible intruders.

5.2 Procedure

The study included three meetings with participants, on the first, second and ninth days, with the following structure:

Day 1

Step 1: Briefing Participants were explained system's concept and functionality, the purpose of the study and the procedure, and asked for consent to proceed.

Step 2: Initial Interview Semi-structured interviews to identify security measures used by participants and also their privacy concerns regarding mobile devices.

Step 3: Installation and Enrollment Installation of our application in the participant's own device and initial set up. The enrollment was conducted to detect usability issues, since the version installed didn't perform face matching.

Day 2

Step 1: Interview Semi-structured interviews to identify changes in behavior, to assess if participants remembered their interactions, and their expectations regarding unauthorized access.

Step 2: Sessions Review The logs were reviewed by participants, and contrasted with their answers in the previous step.

Day 9

Step 1: Comments Participants were asked to offer general comments about the application concept and experience as a participant.

Step 2: Exit Interview Semi-structured interviews to summarize how participants adopted the system, and how they changed their behavior or perceptions, if at all.

5.3 Analysis

We recorded and then transcribed audio of the interviews. The analysis of the interviews was done using thematic coding inductively. [4]. The researcher who transcribed the interviews created an initial set of codes. That researcher and another coded two interviews each, compared the results, and agreed on an extended set of codes. The researchers then re-coded the interviews and measured reliability, which found to be acceptable (Cohen's $\kappa = .95$). One researcher coded the remaining interviews. The analysis is based on that researcher's coding.

5.4 Results

Use by third parties We again examined what motivated participants to share their mobile devices, their defenses to protect personal data and level of concern about a third-party using their device.

All participants reported that they have shared their devices with someone else to perform specific tasks: to show something (7/10) such as a photographs, to make a phone call (5/10), to play games (4/10), to send a text message (4/10), to surf the internet (3/10) and others (2/10).

Reported defenses included not handing over the device when showing contents (3/10); keeping supervision when someone is using their device (2/10); keeping the device around (8/10), trusting that friends and family will not snoop through device contents (6/10); and not storing very sensitive information on devices (4/10).

Regarding worries about having someone looking through their device data, 5/10 participants reported that it depended on the type of person or their trust relationship; and 2/10 that they were concerned over anyone using their devices. Only 1 participant wasn't concerned at all.

Bring your own device Two participants reported working in organizations that adopted BYOD policies. They use their own devices to handle professional e-mail and documents. Hence, their concerns were not only over personal data, but also sensitive work products.

> "This [device] has information about the two parts of my life, my personal and professional life." (P8)

Experiences of unauthorized use Two participants shared that someone used their devices without authorization, prior to the study. P9 reported a suspicion that a colleague snooped through her device after sharing it for a phone call. P6 reported that her tablet was accessed without permission to consult specific data.

Impact of participation As a result of participating in study, two participants reported an increased awareness of the threat, and plans to act on it. Before the field study, P9 did not use unlock authentication because it was inconvenient, but afterwards decided to set it up. P2 said that this study helped her realize the sensitive data stored on her device, and that she would now set unlock authentication also.

Adoption Only one participant reported adopting our system as a deterrent. P9 indicated that she informed her family that this kind of application was installed on her device in a way to discourage them to use her phone. Eight participants used the application to discover misbehavior by others, but without any changing their own behavior. One participant used the application for entrapment, i.e. actively creating situations where others could be caught misbehaving.

Problems All but one participant had some kind of difficulty setting up the application, including problems with enrollment, or with connecting the accessibility service or runtime service. These were usability problems that can be easily overcome by creating a wizard to help users through the initial configurations steps.

Half the participants expressed concerns over negative impact on battery life.

Seven participants had some kind of difficulty in interpreting the logs collected by the application. Some participants initially didn't understand the meaning of some the applications that appeared on the logs, such as home, lock screen, and launcher, because these packages are not commonly seen as being apps. This usability problem was mitigated early, and users received an update where packages related with system activities were filtered out.

Four participants reported difficulties in determining if the logged activities were their own. This problem was expected in the instrumented version used in the field study, which doesn't capture pictures because of ethics concerns. Indeed, 9/10 participants reported that they would prefer seeing pictures taken with the front-facing camera along with the logs.

Four participants expressed difficulty in understanding some labels in the app, such as "session", which we meant as the period of time between the device waking up and being again turned off. The main reason why this happened was

because users do not represent their usage as a set of sessions, but as continuous. This issue warrants further evaluation of design alternatives, for instance presenting a condensed time-line, with expandable logs for whole days.

Advantages Four participants indicated having a stronger sense of security with our application.

> "I feel more secure because if someone uses it I will know." (P4)
> "The icon makes me feel more secure, since everybody can see that is being recorded." (P7)

Three participants indicated that the application could be used to monitor a child's activity.

> "I could use it to know what my son does on his tablet" (P6)

The majority of participants (8/10) manifested being pleased with having passive security, e.g. that the app does not require attention, or that it runs on the background.

> "I think it is a type of application that does not require attention. It is running and when you feel the need you see the recordings. I think that's positive." (P5)

> "It doesn't ask me for a PIN or a sketch to use my phone."(P7)

Eight participants were pleased that the application allows them to know *who* used the device without permission, thus confirming our observation in the first study, that users want to closely regulate their trust relationships.

Four participants were pleased that the application allows knowing *what* was done in the device by others, which indicates a type of adoption that focuses more on damage control, as do traditional intrusion detection systems.

6 Limitations

Face Recognition Accuracy One known limitation of face recognition is that it's accuracy strongly depends on light conditions, camera quality and framing of the pictures. In this system, the lack of accuracy can somewhat be mitigated by the fact that multiple pictures are taken. Furthermore, since a false positive will only produce an additional log, the recognition algorithm can be optimized to minimize false negatives. Since the objective of our studies was to assess the feasibility of mobile intrusion detection systems, and how they would be adopted, we leave further investigation into specific biometric techniques (face or otherwise) for future work.

Privacy Implications As much as device owners see the potential in our application to protect their privacy, their friends and family might see it as infringing on their own privacy. In fact, our software could be used as an offensive tool, and the owner might seek to share the device in order to see what other people do, which may include accessing their own accounts. The fact that 3 participants in the field study suggested that our system could be used to control children's activities is revealing of the possibility of misuse. However, we find that if someone's intent were to spy on others, there are several tools better suited for the job than our software, and there's nothing to stop someone from installing spyware on their own devices.

Performance Because we wanted to examine feasibility and adoption issues, we did not conduct a formal performance analysis, nor did we optimize the software, leaving that for future work. Given the possible impacts on battery and data storage, we conducted only a test in a heavy user's device during 8h, between 5pm and 1am. The device was a Wiko Getaway smartphone with Android 4.4. Our system was configured to take pictures every fifteen seconds, while the device was being used, and to record all interactions regardless of the recognition result. In that period, our system was responsible for 5% of battery consumption, and occupied 140MB of storage. The space occupied by the application is essentially due to the size and number of photographs taken during monitoring. The impact on storage is significant and it could be improved by compressing pictures and/or offloading them to the cloud. The impact of battery consumption was not significant but could still be improved.

7 Conclusion

Our approach offers intrusion detection and response capabilities to end-users, specifically for the risk of physical intrusion by socially-close adversaries. Our system can act as a deterrent, and also as a tool for incident management.

In two user studies, we found that users are indeed concerned about third-parties looking through their mobile device data, and that those concerns are often dependent on trust relationships. Despite that, a large number of participants did not use unlock authentication because it is inconvenient or tedious. Our approach bridges this gap, catering to users' desire to protect the contents of their mobile devices, but without having to expend additional effort. Our system offers users the opportunity to know if their phone was attacked, to know who was the attacker, and what they did.

Future work on our implementation will target the usability issues that were identified, on improving performance, and on improving the accuracy of facial recognition by implementing an algorithm based on face and eye detection.

8 Acknowledgements

This work was supported by FCT through funding of the LaSIGE Research Unit (UID/CEC/00408/2013), and a PhD studentship (SFRH/BD/98527/2013).

References

1. S. Audet. JavaCV - Java interface to OpenCV and more. https://code.google.com/p/javacv/. 2009.
2. A. J. Aviv, K. Gibson, E. Mossop, M. Blaze, and J. M. Smith. Smudge Attacks on Smartphone Touch Screens. In Proceedings of the 4th USENIX Conference on Offensive Technologies, WOOT'10, pages 1-7, Berkeley, CA, USA, 2010. USENIX Association.
3. G. Bradski. The OpenCV Library. Dr. Dobb's Journal of Software Tools, 2000.
4. K. Charmaz. Constructing grounded theory. Sage, 2014.
5. E. Chin, A. P. Felt, V. Sekar, and D. Wagner. Measuring user confidence in smartphone security and privacy. Proceedings of the Eighth Symposium on Usable Privacy and Security - SOUPS '12, (1):1, July 2012.
6. H. Crawford, K. Renaud, and T. Storer.: A framework for continuous, transparent mobile device authentication. Computers & Security, 39, Part B(0):127-136, 2013.
7. S. Egelman, S. Jain, R. S. Portnoff, K. Liao, S. Consolvo, and D. Wagner. Are You Ready to Lock? Understanding User Motivations for Smartphone Locking Behaviors. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, pages 750-761, New York, NY, USA, 2014. ACM.
8. A. Hang, E. von Zezschwitz, Alexander De Luca, Heinrich Hussmann. FaceProfiles: Inconspicuous, Private and Secure Mobile Device Sharing Workshop on Inconspicuous Interaction at CHI 2014. Toronto, Canada, April 26- May 1 2014.
9. E. Hayashi, S. Das, S. Amini, J. Hong, and I. Oakley. CASA: Context-aware Scalable Authentication. In Proceedings of the Ninth Symposium on Usable Privacy and Security, SOUPS '13, pages 3:1-3:10, New York, NY, USA, 2013. ACM.
10. IBM BYOD - Bring Your Own Device - United States. http://www.ibm.com/mobilefirst/us/en/bring-your-own-device/byod.html. 2015. [Online; accessed 27-April-2015].
11. H. Khan, A. Atwater, and U. Hengartner. Itus: An Implicit Authentication Framework for Android. Proc. of 20th Annual International Conference on Mobile Computing and Networking (MobiCom 2014), Maui HI, September 2014.
12. A. K. Karlson, A.J. Bernheim Brush, and Stuart Schechter. Can I Borrow Your Phone? Understanding Concerns When Sharing Mobile Phones. In Proceedings of the 27th international conference on Human factors in computing systems - CHI'09, New York, New York, USA, April 2009. ACM Press.
13. I. Muslukhov, Y. Boshmaf, C. Kuo, J. Lester, and K. Beznosov. Understanding users' requirements for data protection in smartphones. In Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops, ICDEW '12, pages 228–235, Washington, DC, USA, 2012. IEEE Computer Society.
14. M. E. Whitman and H. J. Mattord. Principles of Information Security. Course Technology Press, Boston, MA, United States, 4th edition, 2011.
15. E. von Zezschwitz, P. Dunphy, and A. De Luca. Patterns in the Wild: A Field Study of the Usability of Pattern and PIN-based Authentication on Smartphones. In Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services - MobileHCI '13, page 261, New York, New York, USA, 2013. ACM Press.

FBL - Filtro de Bloom Linear

Rui Lima[1] , Carlos Baquero[2] * e Hugo Miranda[3]

[1] CLEGI, Universidade Lusíada - Norte, V. N. Famalicão
rml[at]fam.ulusiada.pt
[2] HASLab / INESC tec, Universidade do Minho, Braga
cbm[at]di.uminho.pt
[3] LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal
hmiranda[at]di.fc.ul.pt

Resumo As estruturas de dados que permitem o armazenamento de informação de forma probabilística (em particular, os filtros de Bloom) caracterizam-se por permitir a regulação do equilíbrio entre a eficiência na gestão do espaço de armazenamento e a precisão das respostas. Esta possibilidade tem motivado a sua utilização em cenários adversos, por exemplo em redes de sensores, onde os dispositivos apresentam recursos limitados (memória, cpu, energia). Este artigo apresenta um mecanismo de Filtro de Bloom Linear (FBL), que permite associar a cada um dos elementos uma probabilidade quantizada no intervalo real [0,1], ultrapassando as limitações impostas pela característica binária dos filtros de Bloom. Os resultados mostram que é possível parametrizar um FBL de modo a manter um erro aceitável em função da variação do número de bits usados na quantização e do número de funções de *hash* usadas na indexação. O artigo discute a aplicação dos FBLs em mecanismos de disseminação e descoberta de recursos em redes de sensores, mostrando como contribuem para manter uma dimensão constante das mensagens trocadas pelos sensores, independentemente da dimensão da rede.

Keywords: Filtros de Bloom, Difusão, Redes de Sensores

1 Introdução

Um Filtro de Bloom (FB) [2] é uma estrutura de dados que permite o armazenamento eficiente de informação de uma forma probabilística. Estas estruturas de dados têm muitas aplicações [16] como por exemplo em dispositivos com baixos recursos de memória [12]. Uma das suas características fundamentais é a elevada eficiência na gestão do espaço de armazenamento de dados. Contudo, a sua natureza não determinística tem sempre associado um grau de incerteza, permitindo que na consulta da informação inserida (interrogação ao filtro) possam surgir erros na resposta, erros estes que se manifestam como falsos positivos. Os FBs

* O trabalho descrito neste artigo foi parcialmente suportado pela FCT - Fundação para a Ciência e Tecnologia, Portugal e desenvolvido no âmbito do projeto UID/EEA/50014/2013 (HASLab - INESC TEC).

permitem ajustar alguns dos seus parâmetros assegurando um limite máximo para o erro esperado, de forma que cada aplicação possa estabelecer um nível de erro que seja comportável.

A motivação imediata para a proposta de um Filtro de Bloom Linear (FBL), resulta da necessidade de gestão eficiente de informação difundida e armazenada para a pesquisa de recursos em redes WSN (Wireless Sensor Network). Atendendo que os sensores podem encaminhar as mensagens uns dos outros (*multi-hop*), os FB podem ser incluídos nas mensagens trocadas entre os sensores, uma aproximação já utilizada em [7,4] para identificar quais os recursos detidos por cada nó da rede. No entanto, a informação mantida pelos FB tradicionais é limitada a uma indicação binária da presença/ausência de cada recurso (elemento do conjunto).

O FBL estende o FB tradicional ao permitir associar um parâmetro adicional a cada elemento que é inserido no filtro. O que permite ao FBL responder a interrogações sobre a presença de um elemento e ainda obter uma indicação sobre o nível de confiança dessa resposta. Uma característica interessante do parâmetro de confiança é a possibilidade de ajustar o seu valor em função dos diferentes nós em que é registado. No caso das redes multi-hop, este valor pode ser usado para inferir sobre a distância a que se encontra um determinado recurso, um aspeto muito relevante para redes sem infraestrutura como as redes de sensores.

O artigo discute o equilíbrio entre a resolução dos parâmetros de configuração e o número de elementos do FBL, demonstrando que a calibração do FBL permite obter um erro aceitável em função da variação do número de bits usados na quantificação e do número de funções de *hash* usadas na indexação. O FBL utiliza uma técnica bastante geral, potencialmente aplicável a outros contextos.

2 Trabalho Relacionado

Os Filtros de Bloom (FB) [2] e as suas variantes têm sido muito usados em vários domínios [16]. Um FB permite representar sucintamente um conjunto, à custa de uma dada probabilidade de falsos positivos ao testar a pertença de elementos nesse conjunto. A versão mais simples de um FB pode ser implementado recorrendo a um simples vetor de elementos binários. Ao inserir um dado elemento, são calculadas k coordenadas no vetor pela aplicação de igual número de funções de *hash*. Os bits nestas coordenadas são passados a 1 para sinalizar a inserção. Dualmente, a interrogação da pertença verifica se todas as coordenadas do elemento a testar estão a 1. A calibração do k ótimo depende no tamanho do vetor e do número de elementos a inserir. Os falsos positivos resultam da possibilidade de um elemento não inserido aparentar estar presente, se as suas coordenadas coincidirem todas com outros elementos inseridos no filtro.

Sendo os FB representações de conjuntos, a necessidade de noutros contextos de aplicação representar multi-conjuntos permitindo remoções, conduziu a uma variante designada por *Counting Bloom Filters* (CBF) [6]. No CBF o vetor deixa de ter valores com um único bit passando a ter um pequeno contador, permitindo operações de eliminação ao incorporar a possibilidade de decrementar o

respetivo contador. O processamento de grandes quantidades de dados necessita de otimizações das operações principais, reduzindo o tempo de execução para a inserção e remoção de elementos [9]. Uma outra forma de otimização consiste em melhorar as funções de *hash*, permitindo aumentar a eficiência de execução e reduzir o espaço de armazenamento [3]. A variante Bloomier [4] permite codificar funções estendendo as tradicionais interrogações para testar se um elemento é membro de um conjunto, ao permitir associar atributos a cada um dos elementos disponíveis. A sua estrutura resulta numa utilização em paralelo de múltiplos FB. Ao ser adicionado a mensagens de pesquisa em redes sem fios, permitiu desenvolver aplicações de aquisição de contexto e descoberta de recursos [16].

Para evitar filtros sobredimensionados surgiu a variante *Scalable Bloom Filters* (SBF) [1] que permite adaptar dinamicamente a qualidade do filtro, sem saber previamente o número de elementos a inserir. Desta forma foi possível ultrapassar uma das principais limitações dos FBs, que consistia na necessidade de definir o tamanho do filtro em função do número de elementos a inserir e da qualidade do filtro. A utilização de FBs em aplicações distribuídas despoletou uma técnica que permite implementar FB de crescimento dinâmico [8], em que a dimensão do filtro é ajustada (durante a execução) em função do número de elementos inseridos.

Ao transmitir um FB em mensagens através da rede é necessário considerar aspetos de segurança, tomando ações simples como por exemplo aumentar a probabilidade de esquecer elementos antigos face a elementos mais recentes, quando existe sobreposição de elementos [11] no filtro. Numa WSN os sensores podem adquirir informação de contexto através da escuta das mensagens dos seus vizinhos. Os FB podem resumir essa informação de contexto para melhorar os protocolos de encaminhamento [10] em redes 6LoWPAN (*IPv6 over Low-Power Wireless Personal Area Networks*). Em vez de processar e armazenar toda a informação das tabelas de encaminhamento sobre redes IP, é possível obter bons resultados apenas com resumos destas tabelas armazenados em FB, recolhendo essa informação recorrendo à troca de mensagens [17]. Começam a surgir propostas alternativas [14] ao FB tradicional, como por exemplo o filtro Cuckoo [5] que permite a remoção de elementos dinamicamente, melhorando a performance do filtro em espaço de armazenamento e tempo de consulta.

A variante *attenuated Bloom filter* [15], foi aplicada num cenário de pesquisa de documentos sobre redes *peer-to-peer*. Este tipo de filtro é formado por uma matriz de FBs, em que na posição i da matriz está armazenado o FB resultante da contribuição de todos os servidores que se encontram à distância i, de modo a estimar a distância em *hops* entre servidores com documentos relacionados. Este mecanismo permite adquirir contexto dos servidores vizinhos de modo a obter um algoritmo de localização probabilístico. Esta abordagem revela um potencial elevado de aplicação em WSNs, mas o facto da dimensão desta estrutura de dados depender do diâmetro da rede, é uma condição demasiado exigente para sensores com recursos (memoria, cpu, energia) limitados. Existe ainda um mecanismo de encaminhamento que propõem o decaimento da informação contida num FB, para encontrar um gradiente que corresponda à rota entre dispositivos numa

rede WSNs [7]. Contudo o mecanismo de difusão é demasiado pesado e não prevê a sua aplicação de forma eficiente em redes com topologia dinâmica.

3 Filtro de Bloom Linear

Face a estas limitações, este artigo vem propor um Filtro de Bloom Linear (FBL), que inspirado nas características das variantes anteriores, permite atender às limitações impostas pelos sensores mantendo uma dimensão constante da estrutura de dados.

3.1 Filtro de Bloom

A estrutura de dados mais simples para um FB é a sua implementação como um vetor binário, onde cada posição pode armazenar um único bit $\{0, 1\}$. No FB a informação é armazenada com uma técnica probabilística, para permitir operar sobre grandes quantidades de dados, fazendo uma gestão eficiente do espaço de memória disponível. A dimensão do filtro é representada por m e corresponde ao número de posições disponíveis no vetor binário. O número de elementos inseridos no FB é representado por n.

Sobre um FB podem-se realizar duas operações principais: **inserir** (armazenar elementos no filtro) e **interrogar** (questionar o filtro para saber se este contém um determinado elemento). A operação de inserção de elementos corresponde a atribuir a cada uma das k posições do vetor (obtidas usando k funções de *hash* independentes) o valor 1. A operação de interrogação começa por aplicar as k funções de *hash* aos elementos, retornando uma resposta afirmativa quando verifica que todas essas k posições do vetor tem o seu valor definido a 1. A natureza probabilística do FB permite falsos positivos, ou seja o resultado da interrogação pode devolver uma indicação errada sobre a pertença de um elemento, embora esse elemento nunca tenha sido inserido. A probabilidade de falsos positivos é dada por p^k, sendo p a probabilidade de um determinado bit não ser 1. A análise combinatória resulta que após a inserção de n elementos o valor de p pode ser calculado pela Eq.(1).

$$p = \left(1 - (1 - 1/m)^{nk}\right) \approx \left(1 - e^{-nk/m}\right) \tag{1}$$

É possível calcular analiticamente o valor de k que minimiza a probabilidade de falsos positivos usando a Eq.(2) (mas na prática usa-se sempre k inteiro). A probabilidade de falsos positivos f é dada pela Eq.(3).

$$k = \frac{m}{n} \ln 2 \tag{2}$$

$$f \approx (1/2)^k \approx (0.6185)^{m/n} \tag{3}$$

As equações (2) e (3) permitem escolher k em função de f, contudo é possível enunciar também uma **regra prática** para o dimensionamento de um filtro: Caso

se conheça a ordem de grandeza dos elementos a inserir no filtro, o FB deve ter uma dimensão $m \approx 10 \times n$ e para assegurar uma probabilidade de falsos positivos que seja inferior a 1%, então resulta $k = 7$.

Inicialmente os filtros arrancam com todas as posições a zero. A Figura 1 representa a fase de inserção dos elementos $\{a, e\}$, sobre um FB previamente carregado com os elementos $\{b, c, d\}$, usando $k = 4$ funções de hash_k independentes para gerar os índices do vetor.

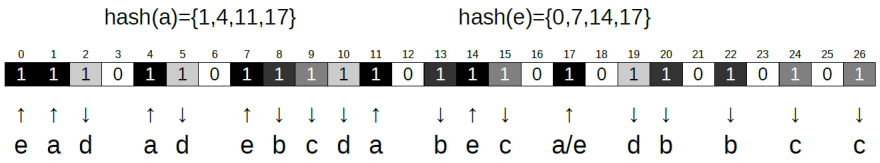


Figura 1: FB com os elementos {a,b,c,d,e}

Ao inserir $\{a\}$ as posições $\{1, 4, 11, 17\}$ são colocadas a 1, mas ao inserir o elemento $\{e\}$ nas posições $\{0, 7, 14, 17\}$ surge uma sobreposição, que por si só não perturba o resultado da interrogação. Contudo, dada a densidade do filtro apresentado na Figura 1, caso sejam inseridos mais elementos o filtro tende a saturar. Neste caso, a taxa de falsos positivos aumenta até ao ponto em que o filtro fica completamente saturado, ficando o vetor correspondente com todas as posições definidas a 1 e o filtro deixa de ter utilidade. Estudos anteriores [15,3,4] exploraram este efeito, de modo a ter uma regulação do equilíbrio entre a eficiência na gestão do espaço de armazenamentos e a precisão das respostas, concluindo que o número de bits a 1 deve ser aproximadamente metade da dimensão do filtro.

3.2 FBL: Estrutura e Operadores

O Filtro de Bloom Linear (FBL) é uma estrutura semelhante a um Filtro de Bloom que permite armazenar junto a cada elemento uma grandeza totalmente ordenada (dando um resultado único perante as funções binárias *max* e *min*). Assim, já não se trata de representar um conjunto, mas sim uma associação que faz o mapeamento entre elementos e grandezas numéricas. Este artigo considera uma representação das grandezas numéricas em números reais limitados ao intervalo $[0, 1]$. A cada elemento fica associado um parâmetro que indica o grau de confiança c com que o elemento é adicionado ao conjunto. Por omissão o FBL é inicializado com o valor 0 em todas as posições do vetor (tal como usual, 0 indica a não pertença). Caso seja usado $c = 1$ para todos os elementos inseridos, então o FBL comporta-se como um FB tradicional. Nos restantes casos é possível considerar um grau de incerteza considerando valores intermédios para o parâmetro de confiança $0 < c < 1$. Para associar um grau de confiança a cada elemento inserido no FBL, é formado o par $(elem_n, c_n)$. A Figura 2 ilustra uma inserção análoga à apresentada na Figura 1 mas realizada sobre um FBL.

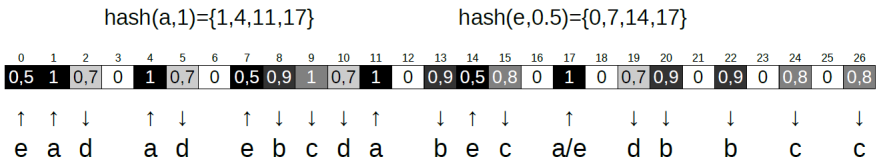


Figura 2: FBL com os elementos {a,b,c,d,e} e respetivo parâmetro de confiança

A operação de **interrogação** ao FBL é realizada pela função *conFBL*(*elem*) apresentada no Alg. 1. Caso a consulta retorne $c = 0$, então *elem* não faz parte do conjunto armazenado no FBL. Caso a consulta retorne $c \in]0, 1]$, então o filtro funciona como um estimador, retornando o grau de confiança estimado $\hat{c}$ para o elemento *elem*.

A operação *min* da função *conFBL*(*elem*) assegura que em casos de sobreposição parcial, como acontece no FBL apresentado na Figura 2 na posição 17, a consulta do parâmetro c não é afetada. Contudo, o resultado da interrogação pode sofrer uma **sobreestimação de pertença** resultante de um erro de estimação por excesso, caso a sobreposição seja completa (sobreposição em todas funções de *hash*). A sobreestimação de pertença é uma generalização do conceito de falso positivo no contexto dos FBs, que se deteta quando o estimador devolve um grau de confiança c superior ao inserido para um dado elemento, ou seja $\hat{c} > c$. Este comportamento pode resultar de um mau dimensionamento do filtro, podendo ser um indicador da entrada em saturação do FBL.

Algorithm 1: Operações sobre FBL

```
1  const FBL[] ;                                   // Filtro de Bloom Linear
2  const K ;                        // Número de funções de hash independentes
3  Function conFBL (elem)
4  |  c ← 1;
5  |  for i ← 1 to K do
6  |  |  idx ← hash_i(elem);
7  |  |  c ← min (c,FBL[idx]);
8  |  end
9  |  return c;
10 Function insFBL (elem,c)
11 |  for i ← 1 to K do
12 |  |  idx ← hash_i(elem);
13 |  |  FBL[idx] ← max (c,FBL[idx]);
14 |  end
```

A operação de **inserção** é semelhante à realizado num FB tradicional, mas em vez se armazenar um único bit para cada uma das k posições do vetor, o valor c é adicionado por uma operação de majoração (*max*) em cada uma das k

células do FBL. A processo de inserção é realizada pela função $insFBL(elem, c)$ apresentada no Alg. 1. A Figura 2 apresenta um exemplo de um FBL sobre o qual estão a ser inseridos os elementos $\{(a, 1); (e, 0.5)\}$, através das operações $insFBL(a, 1)$ e $insFBL(e, 0.5)$ respetivamente.

Exemplo de aplicação: Uma operação de **atenuação**, que pode ser invocada por sensores de uma rede WSN, pode multiplicar todas as células do filtro por um dado fator (e.g. 0.9). O efeito é reduzir o grau de confiança associado a todos os elementos presentes no filtro, que no limite se aproxima de 0. O uso desta operação sobre a receção de FBL difundidos pelos sensores vizinhos, permite criar localmente gradientes sobre a localização de recursos em redes multihop [13].

3.3 FBL: Quantização Linear

Havendo um número infinito de reais em qualquer segmento de reais, em termos de implementação num FBL, seria necessário armazenar um número infinito de bits para fazer a representação binária de c. Existem várias formas para representar um número de vírgula flutuante na sua forma binária, como por exemplo a norma IEEE 754. Esta norma define os formatos adequados para representar números de vírgula flutuante com vários níveis de precisão, sendo os mais comuns: i) precisão simples (32 bits) e ii) precisão dupla (64 bits). O FBL não impõe nenhuma restrição sobre a técnica a ser utilizada para fazer a representação binária e respetiva compressão de um número de vírgula flutuante. Contudo, sugere-se a utilização de um mecanismo de quantização simples que permita a atribuição de valores discretos para uma grandeza contínua. Atendendo que no FBL o parâmetro a representar é $c \in [0, 1]$, são assumidas algumas simplificações para fazer a respetiva representação binária. Como c não tem sinal nem expoente, é suficiente fazer a representação binária da sua mantissa. A norma IEEE 754 de precisão simples aloca 23 bits para a representação da mantissa. Atendendo que o FBL poderá ser utilizado em redes de sensores, não faz sentido manter uma mantissa com esta precisão porque: i)o FBL é uma estrutura de dados probabilística (com incerteza); ii) os dispositivos sensores tem significativas restrições de memória e CPU.

O número de bits b usados no processo de quantização será um dos principais parâmetros de calibração do FBL. O espaço de memória alocado por cada FBL, i.e. a dimensão de FBL é $m \times b$ bits, sendo m número máximo de células disponíveis no FBL e b o número de bits usados em cada uma das células. Estes parâmetros devem ser ajustados em função da precisão aceitável pela aplicação final, sabendo que à medida que b aumenta o erro de quantização diminui, mas em contrapartida, aumenta linearmente a dimensão do FBL.

Sabendo que o maior número inteiro positivo representado por um conjunto b bits é (2^b-1), então é possível quantificar linearmente c dividindo o intervalo $[0, 1]$ em partes iguais, ou seja: $c \times (2^b-1)$. Contudo, o resultado desta operação poderá ser ainda um número real, que por uma operação de arredondamento pode ser convertido num inteiro, assumindo um erro máximo absoluto de quantização (FBL_{ERR}) dado pela Eq.(4).

$$FBL_{ERR} = \pm 1/(2^b - 1) \tag{4}$$

O arredondamento poderá ser parametrizado em função das necessidades das aplicações, mas considerando que tipicamente aplicações sobre WSNs associam $c = 1$ ao facto do recurso ser local ao sensor e que isso deve corresponder a ter todos os bits das respetivas células a "1", então é preferível fazer uma operação de truncatura implementada pela função $floor()$, para extrair apenas a parte inteira. Desta forma assegura-se que se existir uma atenuação, por mais pequena que seja, a representação binária nunca terá todos os bits a "1". A representação binária do parâmetro de confiança c pode ser obtida pela chamada da função $c2bit(c, b)$ apresentada no Alg.(2).

Algorithm 2: Representação Binária de c

```
1 Function c2bit (c,b)
2 |  return floor( c × ((1 ≪ b) − 1) ;        // floor(x) : ⌊x⌋
```

Exemplo: Se escolher uma precisão de $b = 5$ bits, então fica definido o erro absoluto $FBL_{ERR} = \pm 0,032 \approx \pm 3\%$.

A Figura 3 ilustra a representação correspondente às células [11-14] da Figura 2, onde é possível observar como o parâmetro confiança c é armazenado no filtro FBL, de modo que este seja formado por um vetor de bits, com uma dimensão controlada.

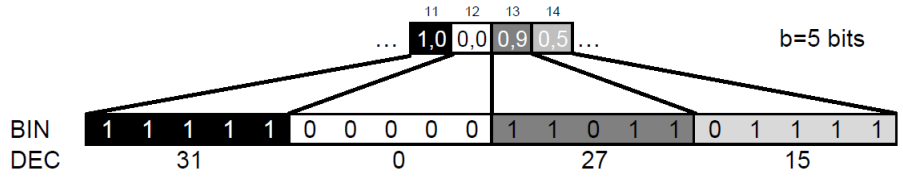


Figura 3: FBL - Representação Final em Memória

O FBL pode ser representado por um vetor de células, contendo em cada célula a representação binária do parâmetro de confiança c, como ilustrado na Figura 3.

Para interpretar a informação armazenada no FBL é necessário recorrer a operações inversas às anteriormente descritas. Por exemplo, para que um sensor possa realizar uma operação de atenuação, será necessário interrogar o FBL para obter a representação binária do valor c, posteriormente aplica-se uma operação inversa da quantização para obter a representação do valor estimado na forma de um número real, e por último aplicar a operação de atenuação.

4 Avaliação

Atendendo ao facto do FBL ser uma estrutura de dados probabilística, a interrogação ao FBL pode devolver uma informação distinta da inserida. Esta avaliação permite caracterizar como se comporta o FBL na sua função de **estimador** $\hat{c}$ correspondente ao mapeamento dos seus elementos noutra grandeza numérica. A avaliação mostra como se comporta o FBL em função da calibração dos principais parâmetros de configuração. No processo de avaliação são mantidas duas estruturas de dados para armazenar o mapeamento $(elem_n, c_n)$. Uma determinística que utiliza um vetor (*vetorElem[]*) que associa c a cada um dos $elem$ e outra implementada por um FBL que permite estimar o valor de c, ou seja $\hat{c}$. O comportamento médio do FBL é avaliado por um conjunto de testes correspondentes a $N = 100000$ iterações num ambiente de simulação em computador.

Os principais parâmetros do ambiente de simulação são apresentados na Tabela 1. Em cada iteração são gerados n pares $(elem_n, c_n)$, em que $elem_n$ é um identificador sequencial e c_n atribuído por uma distribuição pseudo-aleatória de valores uniformemente distribuídos no intervalo $[0, 1]$. Esta informação de mapeamento associativo é armazenada nas duas estruturas; i) determinística (*vetorElem[]*) e ii) FBL. A avaliação realizada tem em conta apenas os verdadeiros positivos, pois os elementos inseridos/consultados fazem parte do conjunto disponível no *vetorElem[]*.

Tabela 1: Parâmetros do Ambiente de Simulação

Atributo	**Valor/Expressão**
N (nº de iterações)	100000
dim(FBL)	4096 bits
m (células)	$\frac{dim(FBL)}{b}$
Erro de quantização (FBL_{ERR})	$\pm 1/(2^b - 1)$
4.2) Precisão (b)	1 bit $\leq b \leq$ 16 bits
4.3) Nº de funções de *hash* (k)	$2 \leq k \leq 16$

Nas subsecções 4.2 e 4.3 estão subjacentes as seguintes **métricas** para caracterizar os resultados obtidos com $N = 100000$ iterações:

- **Valor Médio** $\overline{\hat{c}}$ para o parâmetro confiança c, ver Eq.(5)
- **Desvio Padrão** σ para o parâmetro confiança c, ver Eq.(6)
- **Erro Quadrático Médio** do estimador $EQM(\hat{c})$, correspondente à média dos erros quadráticos, ver Eq.(7)
- **Taxa de ocupação** do filtro, correspondente à razão entre o número de bits a 1 e $(m \times b)$.

Equações das métricas estatísticas aplicadas aos resultados das simulações e utilizadas na avaliação do FBL:

$$\bar{\hat{c}} = \frac{1}{N \cdot n} \sum_{i=1}^{N \cdot n} \hat{c} \tag{5}$$

$$\sigma = \sqrt{\frac{1}{N \cdot n} \sum_{i=1}^{N \cdot n} [\hat{c} - \bar{\hat{c}}]^2} \tag{6}$$

$$EQM(\hat{c}) = \frac{1}{N \cdot n} \sum_{i=1}^{N \cdot n} [\hat{c} - c]^2 \tag{7}$$

Para implementar o FBL é necessário escolher um valor para k, que deve ser ajustado em função do ambiente de simulação (neste caso parametrizado na Tabela 1). Para obter uma aproximação teórica para o valor de k, assumiu-se que o número de elementos inseridos no FBL não ultrapassa os 100 elementos ($n = 100$) e que uma precisão de $b = 4$ bits é suficiente para representar c, então de acordo com a Eq.(2) obtém-se $k = \frac{4096/4}{100} \ln 2; k \approx 7$.

4.1 Impacto da precisão na capacidade do FBL

Como se pretende manter a dimensão do FBL (dim(FBL)) num valor fixo, o número de células disponíveis no FBL vai depender do número de bits usados em cada célula (precisão), ou seja dim(FBL)=$m \times b$.

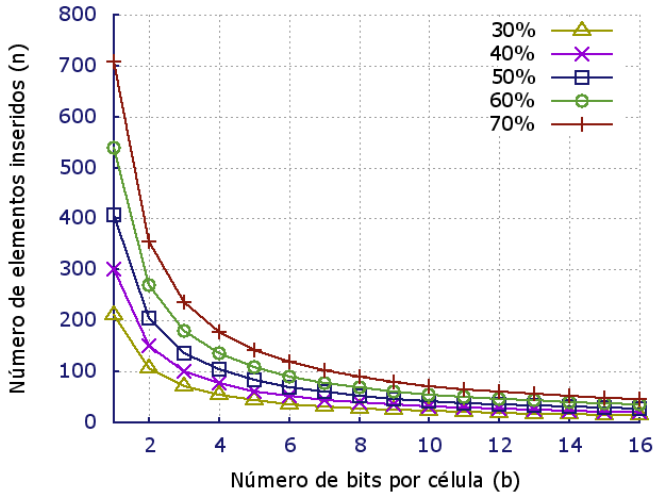


Figura 4: Impacto do nº de bits de precisão na capacidade do FBL

A Figura 4 compara diferentes cenários de carga considerando um FBL com $k = 7$ funções de *hash*, de acordo com a aproximação teórica apresentada acima. Neste conjunto de cenários a taxa de ocupação do FBL é mantida constante em valores pré-definidos entre 30% e 70%.

O processo utilizado para determinar o número de elementos a inserir para cada uma das taxas de ocupação em função dos número de bits de precisão, consistiu em inserir elementos ao FBL e parar quando a taxa de ocupação pretendida é atingida. Este processo de contagem foi repetido 1000 vezes, de modo a obter o inteiro mais próximo do valor médio dos elementos inseridos.

Na Figura 4 é possível observar a capacidade do FBL, correspondente à evolução do número de elementos inseridos no filtro em função do aumento de bits de precisão, garantindo uma taxa de ocupação constante.

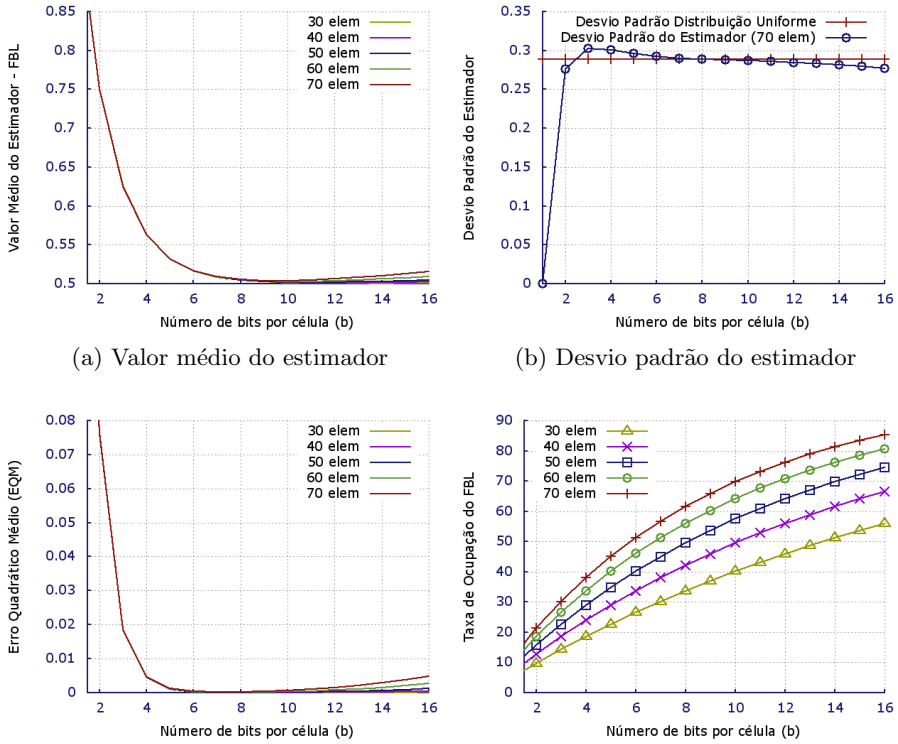
Como previsível, o número de elementos armazenado no FBL aumenta à medida que aumenta a percentagem de ocupação. Contudo, o diferencial entre taxas de ocupação é atenuado à medida que os elementos aumentam a sua precisão. Para armazenar cerca de 100 elementos no FBL e manter uma taxa de ocupação de 50% (Figura 4), o número máximo de bits para a quantização não deve ultrapassar os 4 bits por célula. O número de bits por célula tem um grande impacto na capacidade de armazenamento de dados num FBL de dimensão fixa, em especial na fase inicial do aumento da precisão. Analisando o gráfico apresentado na Figura 4 é possível identificar a ordem de grandeza da capacidade de armazenamento do FBL, que será usada durante as restantes subsecções da avaliação, conjuntamente com as parametrizações apresentadas na Tabela 1.

4.2 Impacto da calibração da precisão (b) no FBL

Para os testes de avaliação considerou-se constante o parâmetro $k = 7$ (ver Eq.(2)), variando o número de elementos inseridos (n) em função da precisão das células (b).

Para avaliar o comportamento do filtro em função da precisão b, são considerados cinco cenários correspondentes à inserção de 30, 40, 50, 60 e 70 elementos, comparando casos com taxas de ocupação reduzidas até zonas próximas da saturação. A Figura 5 apresenta quatro gráficos para caracterizar estatisticamente o estimador $\hat{c}$ proporcionado pelo FBL. Sabendo que c assume valores de uma distribuição uniforme no intervalo $[0, 1]$, o valor médio esperado é 0.5. Este valor foi comprovado durante a avaliação, pois o valor médio calculado com a informação armazenada estrutura determinística foi de $\overline{c} = 0.5005$. Para minimizar a perturbação causada por algum evento/característica que possa surgir durante a comparação da qualidade do estimador, foi escolhido o valor médio como indicador estatístico do estimador, ou seja $\overline{\hat{c}}$.

Na Figura 5a é possível observar como o $\overline{\hat{c}}$ se afasta desse valor ideal (linha base 0.5) para valores demasiado baixos de b, provocando erros significativos na quantização e induzindo o FBL a uma sobreestimação exacerbada. Para caracterizar a dispersão de valores face à média determinou-se o respetivo desvio padrão usando a Eq.(6), que para facilitar a leitura da Figura 5b apenas se apresenta



(a) Valor médio do estimador

(b) Desvio padrão do estimador

(c) Erro Quadrático Médio do estimador

(d) Taxa de ocupação do FBL

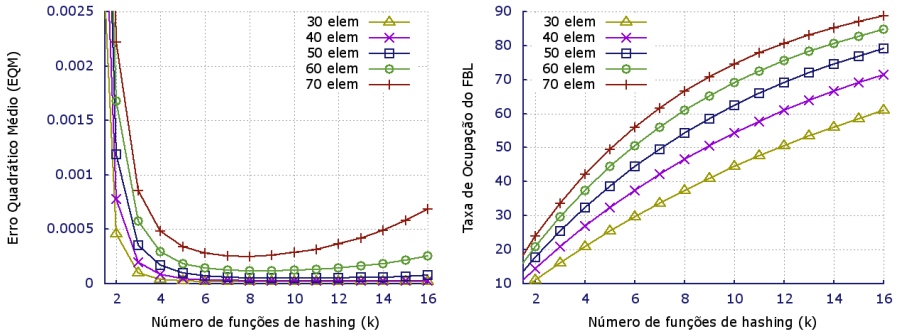
Figura 5: FBL - Influência da precisão (b) na calibração do estimador

o caso correspondente à inserção de 70 elementos. A comparação apresentada na Figura 5b permite confirmar que a dispersão do estimador é semelhante à dispersão característica de uma distribuição uniforme, exceto no caso em que $b = 1$ (que por usar apenas um único bit tem dispersão nula).

Para realizar a representação binária de c (no caso estudado) seriam suficientes 5 bits por célula. A Figura 5d permite observar a taxa de ocupação do filtro em função do número de elementos inseridos e do número de bits por célula. Na Figura 5c é possível observar que o erro do estimador se aproxima de zero quando $5 \leq b \leq 12$, ou seja existe um limite inferior mínimo e um limite máximo aconselhável para o número de bits por célula. A análise conjunta das Figuras 5d e 5c, permite concluir nos casos em estudo que: i) o erro com b inferior a 5 é originado por erros de quantização já que a taxa de ocupação do filtro é inferior a 50%; ii) quando b é superior a 12 a origem do erro resulta de uma utilização demasiado densa do filtro, pois a taxa de ocupação do filtro é superior a 50%, aproximando-se claramente da zona de saturação.

4.3 Impacto do número das funções de *hash* (k) no FBL

Para completar a avaliação do FBL foram considerados cinco cenários com uma variação dos elementos inseridos n semelhante à avaliação anterior, mas fixando uma precisão de $b = 8$ bits por célula (limitando o erro absoluto de quantização a $\pm 0,004$) e variando $2 \leq k \leq 16$. Estes cenários permitem avaliar o impacto que o número de funções de *hash* exercem sobre o valor médio do estimador, obtido ao interrogar o FBL.



(a) Erro Quadrático Médio do estimador (b) Taxa de ocupação do FBL

Figura 6: FBL - Influência do número de funções de *hash* (k) na calibração do estimador

Para o cenário com $n = 70$ elementos inseridos no filtro (com 100000 iterações) obteve-se $\overline{\hat{c}} = 0.5074 \pm 0.0051$ e $\sigma = 0.2882 \pm 0.0007$, ou seja o FBL não se afasta dos parâmetros típicos de uma distribuição uniforme no intervalo $x \in [0, 1]$, em que $\overline{x} = 0.5$ e $\sigma_x \simeq 0.2887$. Ao variar k o valor médio do estimador e o respetivo desvio padrão permaneceram praticamente constantes, ou seja a resposta do filtro não apresenta oscilações significativas para a média e desvio padrão.

A imagem da Figura 6a mostra claramente que o intervalo $4 \leq k \leq 12$ corresponde à zona de menor impacto no estimador, pois os valores do erro $EQM(\hat{c})$ são inferiores a 0.0005. O limite inferior assegura que em casos de sobreposição das funções de indexação, o erro do estimador não seja predominantemente influenciado por um k pequeno. Em contrapartida, não se pode aumentar demasiado o número de funções de *hash*, porque isso faz aumentar o espaço ocupado por cada elemento e atendendo que o FBL tem uma dimensão fixa pode provocar a saturação do FBL. Como se pode observar na Figura 6b para os casos em estudo, quando $k > 12$ o filtro aproxima-se do modo de saturação e o erro $EQM(\hat{c})$ começa a aumentar (comprar com Figura 6a).

A Figura 6a permite comprovar que o valor teórico obtido pela na Eq.(2) corresponde a uma zona onde o Erro Quadrático Médio é mínimo.

5 Conclusões

Este trabalho apresenta uma nova variação de um filtro de Bloom, designado por Filtro de Bloom Linear (FBL) que permite armazenar junto de cada elemento uma grandeza no intervalo [0,1], mantendo uma dimensão constante do filtro.

A motivação para o desenvolvimento do FBL surgiu da necessidade de difundir informação em redes WSN, mas esta nova estrutura de dados probabilística tem um carácter mais geral, podendo ser aplicável noutras áreas distintas.

A avaliação faz uma análise de sensibilidade dos parâmetros de configuração do FBL, assegurando que o erro do estimador esteja perfeitamente delimitado por comparação com a resposta de uma estrutura determinística. O estudo da calibração do b e do k permitiu encontrar fronteiras para estes parâmetros e caracterizar a técnica do FBL.

Referências

1. P. S. Almeida, C. Baquero, N. Preguica, and D. Hutchison. Scalable bloom filters. *Information Processing Letters*, 101(6):255 – 261, 2007.
2. B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.
3. F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese. An improved construction for counting bloom filters. In *Proceedings of the 14th Conference on Annual European Symposium - Volume 14*, ESA'06, pages 684–695, London, UK, UK, 2006. Springer-Verlag.
4. B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal. The bloomier filter: An efficient data structure for static support lookup tables. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 30–39, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.
5. B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, pages 75–88, New York, NY, USA, 2014. ACM.
6. L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, June 2000.
7. D. Guo, Y. He, and Y. Liu. On the feasibility of gradient-based data-centric routing using bloom filters. *Parallel and Distributed Systems, IEEE Transactions on*, 25(1):180–190, Jan 2014.
8. D. Guo, J. Wu, H. Chen, Y. Yuan, and X. Luo. The dynamic bloom filters. *Knowledge and Data Engineering, IEEE Transactions on*, 22(1):120–133, Jan 2010.
9. K. Huang, J. Zhang, D. Zhang, G. Xie, K. Salamatian, A. Liu, and W. Li. A multi-partitioning approach to building fast and accurate counting bloom filters. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 1159–1170, May 2013.
10. A. Kalmar, R. Vida, and M. Maliosz. Context-aware addressing in the internet of things using bloom filters. In *Cognitive Infocommunications (CogInfoCom), 2013 IEEE 4th International Conference on*, pages 487–492, Dec 2013.

11. R. P. Laufer, P. B. Velloso, and O. C. M. B. Duarte. A generalized bloom filter to secure distributed network applications. *Comput. Netw.*, 55(8):1804–1819, June 2011.
12. X. Li, J. Wu, and J. Xu. Hint-based routing in wsns using scope decay bloom filters. In *Networking, Architecture, and Storages, 2006. IWNAS '06. International Workshop on*, pages 8 pp.–, 2006.
13. R. Lima, C. Baquero, and H. Miranda. Adaptive broadcast cancellation query mechanism for unstructured networks. In *9th International Conference on Next Generation Mobile Applications, Services and Technologies 2015 (NGMAST'15)*, Cambridge, United Kingdom, Sept. 2015.
14. Y. Qiao, T. Li, and S. Chen. Fast bloom filters and their generalization. *Parallel and Distributed Systems, IEEE Transactions on*, 25(1):93–103, Jan 2014.
15. S. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1248–1257 vol.3, 2002.
16. S. Tarkoma, C. Rothenberg, and E. Lagerspetz. Theory and practice of bloom filters for distributed systems. *Communications Surveys Tutorials, IEEE*, 14(1):131–155, First 2012.
17. T. Yang, G. Xie, R. Duan, X. Sun, and K. Salamatian. Towards practical use of bloom filter based ip lookup in operational network. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–4, May 2014.

Armazenamento Distribuído para Redes de Dispositivos Móveis

Ricardo Monteiro João A. Silva João M. Lourenço Hervé Paulino*
{rad.monteiro, jaa.silva}@campus.fct.unl.pt
{joao.lourenco, herve.paulino}@fct.unl.pt

NOVA Laboratory for Computer Science and Informatics
Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade NOVA de Lisboa, 2829-516 Caparica, Portugal

Resumo Os dispositivos móveis em proximidade geográfica representam um conjunto de recursos inexplorados, tanto em termos de capacidade de processamento como de armazenamento, o que abre caminho para novas aplicações com oportunidades e desafios únicos. Os sistemas atuais de partilha de dados (e. g., fotos, músicas, vídeos) para dispositivos móveis exigem que exista conectividade com a Internet para funcionarem. No entanto, em ambientes onde a conectividade com a Internet não é constante ou de boa qualidade (e. g., eventos desportivos e concertos), ou em locais remotos onde as infraestruturas de rede não existem, é difícil (ou mesmo impossível) partilhar dados entre vários dispositivos móveis. Para resolver este problema, os dispositivos móveis podem formar uma rede ad hoc para compartilhar os seus dados e recursos. Neste artigo propomos um sistema de armazenamento distribuído para partilha de dados entre dispositivos móveis de uso diário, e. g., *smartphones* e *tablets*, usando um mecanismo de *melhor esforço* para garantir persistência e disponibilidade de dados suportando *churn* (entrada e saída inesperada de dispositivos).

Palavras-Chave: Armazenamento Distribuído, Dispositivos Móveis, Redes ad hoc, Peer-to-Peer

1 Introdução

O poder de computação e de armazenamento dos *smartphones* atuais rivaliza com o dos *desktops* de há poucos anos atrás. Tal evolução permite executar num *smartphone* corrente tarefas que anteriormente tinham de ser delegadas em sistemas de maiores dimensões. Adicionalmente, hoje em dia, estes pequenos dispositivos oferecem múltiplas funcionalidades—como tirar fotografias, gravar

* Este trabalho foi parcialmente financiado pela Fundação para a Ciência e Tecnologia (FCT-MEC), no contexto do projeto de investigação CMUP-ERI/FIA/0048/2013, do plano estratégico PEst-UID/CEC/04516/2013 e bolsa de investigação SFRH/BD/99486/2014.

vídeos ou obter sinais GPS—que antes só eram possíveis com dispositivos especializados de difícil acesso para muitos dos consumidores. Dada a proliferação e o aumento das capacidades dos dispositivos móveis, é realista considerar as oportunidades que um conjunto de dispositivos móveis pode colaborativamente apresentar em termos de recursos computacionais e de armazenamento de dados.

Recentemente, tem havido interesse na utilização de uma nuvem de *smartphones* para formar uma rede de armazenamento para aplicações avançadas [7]—por exemplo, uma coleção de oito *smartphones*, cada um com 16 GB de armazenamento, pode representar coletivamente uma nuvem com uma generosa capacidade de 128 GB. Este conceito pode ser particularmente interessante em determinadas situações. Por exemplo, em locais onde não existem infraestruturas de rede disponíveis—localizações remotas, cenários de catástrofe natural, etc. Também em locais com um grande número de pessoas, as infraestruturas existentes podem não ter capacidade para suportar o elevado número de conexões, e. g., eventos desportivos ou concertos. Em eventos particulares, como festas de aniversário, reuniões ou pequenas conferências, também pode não compensar montar uma infraestrutura de rede espeficicamente para esses eventos, devido aos custos elevados ou a configurações demoradas.

Drolia et al. [9] comparam o processamento de dados numa nuvem de *smartphones* contra o envio desses mesmos dados para serem processados numa infraestrutura remota, no que se refere à duração em tempo, ao consumo de bateria e à quantidade de dados transmitidos. Os seus resultados mostram que para algumas classes de aplicações, o processamento na nuvem de *smartphones* é mais eficiente em termos de bateria consumida e latência na obtenção dos resultados.

Armazenamento distribuído em redes móveis ad hoc é um tema com alguma investigação [10,11], mas a maioria dos sistemas existentes não tem uma implementação concreta e funcional para dispositivos móveis ou, no caso específico do iTrust [11], não garante persistência dos dados (i. e., quando um dispositivo se desconecta, os conteúdos partilhados por esse dispositivo ficam inacessíveis).

Neste artigo apresentamos as características e arquitectura de um sistema de armazenamento e partilha de conteúdos distribuído para dispositivos móveis. O sistema permite a publicação de conteúdos privados e a obtenção de conteúdos partilhados por outros utilizadores conetados à mesma rede. Adicionalmente, este tenta garantir a persistência e disponibilidade dos dados na presença de *churn* (entrada e saída inesperada de dispositivos na rede). No entanto, dada a natureza volátil e dinâmica dos ambientes móveis não é possível fornecer uma garantia absoluta.

Desenvolvemos um protótipo para Android onde o utilizador pode partilhar as suas fotografias, visualizar os nomes e *thumbnails* das fotografias publicadas por outros utilizadores e obter essas mesmas fotografias. Este protótipo pode ser facilmente alterado para partilha de vários tipos de conteúdos diferentes (por exemplo, vídeos, músicas, etc.) e funciona em dispositivos móveis existentes, sem que se seja necessário realizar qualquer tipo de alteração ao dispositivo, como a obtenção de permissões de super-utilizador (*root*). A limitação atual passa

pela necessidade de existir uma rede sem fios disponível para os dispositivos se conectarem. Esta não precisa estar conectada à Internet e pode ser fornecida por um dispositivo móvel ativando a funcionalidade *hotspot*.

O resto do artigo segue a seguinte estrutura. Na Secção 2 é apresentado algum contexto, os desafios que se colocam e como foram ultrapassados. De seguida, na Secção 3, apresentada-se a arquitetura do sistema com as suas várias camadas. Na Secção 4 reporta-se e analisa-se resultados experimentais. Na Secção 5 discute-se algum trabalho relacionado. Por último, na Secção 6 apresenta-se as conclusões finais e algum trabalho futuro.

2 Armazenamento de Dados para Dispositivos Móveis

Um sistema de armazenamento para redes ad hoc de dispositivos móveis tem de lidar com diversos desafios inerentes a este tipo de ambiente volátil e dinâmico. A descoberta/localização de recursos neste tipo de ambientes assemelha-se em muito às funcionalidades oferecidas pelos sistemas *peer-to-peer* (P2P) construídos para a Internet. Para além disso, existem os desafios inerentes a ambientes móveis, como a alta mobilidade dos nós (causando mudanças na topologia da rede), as entradas e saídas inesperadas de nós na rede (*churn*), e a comunicação volátil e de qualidade irregular. Tendo em conta os dispositivos móveis, também é necessário lidar com as suas propriedades e limitações características, como a bateria e largura de banda limitadas.

2.1 Localização de Recursos

A localização de recursos é um componente bastante importante em redes ad hoc para dispositivos móveis. Nestas redes a localização de recursos deve ser rápida, de maneira a que as aplicações respondam em tempo útil (e de maneira interativa) aos seus utilizadores, mas também deve ser eficiente, de maneira a haver a menor quantidade de comunicação possível, ajudando na poupança de bateria.

Um mecanismo muito simples para localização de recursos numa rede passa por inundar (*flooding*) a rede com mensagens, perguntando a todos os nós quem é que tem aquilo que se procura. São enviadas mensagens para diversos nós, escolhidos aleatoriamente, até que os recursos procurados sejam encontrados. Este tipo de mecanismo é utilizado em redes não estruturadas, onde os nós não têm conhecimento da localização dos recursos disponíveis. São, também, conhecidos por não serem muito eficientes em redes de grandes dimensões, por isso, é que normalmente são implementados outros mecanismos para prevenir o envio de mensagens em excesso, o que poderia causar a rutura da rede. Um exemplo de um sistema com uma rede não estruturada é o sistema *Gnutella* [16]. O sistema iTrust [11] também usa uma rede não estruturada e um mecanismo de *flooding* para localização de conteúdos.

Pelo contrário, nos sistemas com redes estruturadas, os nós têm uma visão parcial da rede, conhecendo alguns dos seus vizinhos. O tipo de rede estruturada

mais conhecida são as tabelas de dispersão distribuídas (TDD). Estas resolvem o problema do *flooding*, porque empregam um esquema de procura organizada, onde cada nó é responsável por parte dos recursos existentes no sistema. Desta maneira, os nós são capazes de direcionar os pedidos para os nós responsáveis muito mais rápida e facilmente. Um exemplo de uma TDD é o *Chord* [15].

Comparando os apectos positivos e negativos de cada abordagem, decidimos optar pela utilização de uma TDD. Apesar de ter um custo adicional associado à manutenção da rede (o envio de *pings*), acreditamos que num ambiente móvel, onde temos energia e recursos limitados, o custo da manutenção da rede será menor que o custo para garantir persistência de dados numa rede não estruturada. Para garantir persistência de dados numa rede não estruturada, seria necessário a constante verificação dos conteúdos existentes no sistema e da quantidade de réplicas existentes para cada conteúdo. O Phoenix [13] é um exemplo de um sistema com um protocolo de manutenção de réplicas para garantir persistência de dados numa rede não estruturada.

Para além da manutenção das réplicas, com uma TDD resolvemos diversos problemas existentes para atingir o objectivo que pretendemos. A TDD permite-nos determinar e distribuir a responsabilidade dos diversos conteúdos pelos nós existentes no sistema. Desta forma, cada nó é responsável por uma parte dos conteúdos existentes, garantindo que a rede contém, pelo menos, um número mínimo de réplicas para cada conteúdo distribuídas por diferentes nós. Para além disso, a localização e obtenção de conteúdos no sistema por parte de um nó não requer a inundação de pedidos na rede, reduzindo também o consumo de recursos por cada pedido.

2.2 Comunicação

Existem alguns projectos que tentam resolver o problema da criação e operacionalidade de redes ad hoc WiFi compostas por dispositivos móveis (por exemplo: *Serval Project* [2], e *SmartPhone Ad-hoc Networking (SPAN)* [18]). No entanto, ambos os projetos, requerem alterações e configurações nos dispositivos. Para realizar estas modificações é necessário ter permissões de super-utilizador (*root*). Além disso, a gama de modelos suportados por estes projetos é bastante pequena.

Para contornar este problema e criar uma rede ad hoc de múltiplos dispositivos, podemos fazer uso do WiFi Direct [3]. Casetti et al. [5], Lombera et al. [11], e Teófilo et al. [17] usam esta estratégia para os dispositivos comunicarem em modo ad hoc. Outro modo para formar redes ad hoc entre dispositivos móveis é através do Bluetooth [1] (utilizado, por exemplo, em [20]).

Atualmente, o nosso protótipo requer um ponto de acesso (por exemplo: *router*, ou um dispositivo com *hotspot*) e ainda não abordámos a comunicação ad hoc. Como trabalho futuro, o nosso objetivo é estudar e implementar um mecanismo de comunicação ad hoc através de WiFi Direct semelhante ao utilizado por Teófilo et al. [17].

WiFi Direct. O WiFi Direct é um protocolo padronizado pela WiFi Alliance, com o objectivo de permitir comunicações de dispositivo para dispositivo, sem a

necessidade de um ponto de acesso intermédio. A comunicação entre dispositivos usando o WiFi Direct ocorre dentro de um único *grupo*. Um dispositivo do grupo actua como líder (*Group Owner (GO)*) e os restastes dispositivos, designados de *clientes*, são emparelhados ao GO. Os clientes podem ser de dois tipos: *clientes P2P*, têm suporte ao protocolo WiFi Direct; ou, *clientes WiFi*, não têm suporte ao protocolo WiFi Direct e vêem o GO como se fosse um ponto de acesso sem fios convencional.

Os papeis de cada dispositivo dentro do grupo mantêm-se durante toda a sessão. Se o GO se desconectar, os restantes dispositivos têm de criar um novo grupo e eleger um novo GO. Para criar um novo grupo e ser o GO do mesmo, os dispositivos têm que suportar a tecnologia WiFi Direct. Caso contrário, apenas é possível conectarem-se a grupos já formados, vendo o GO como um ponto de acesso. Também é possível a um dispositivo pertencer a mais que um grupo, formando uma ponte entre os dois grupos. Para isso, o dispositivo tem de suportar dois endereços MAC diferentes e existem as seguintes limitações nos dispositivos Android actualmente:

1. Um dispositivo ser GO de mais que um grupo.
2. Um dispositivo ser GO de um grupo e cliente P2P de outro,
3. Um dispositivo ser cliente de mais que um grupo.

Resumindo, o GO de um grupo tem de se conectar como cliente WiFi a outro grupo para formar uma ponte entre os dois grupos. Desta forma, é possível conectar múltiplos grupos e comunicar entre dispositivos de diferentes grupos.

Casetti et al. [5] descrevem esta técnica para dispositivos Android de origem, isto é, sem modificar o hardware ou software original, nem desbloquear o dispositivo para ter permissões de super-utilizador (*root*). Teófilo et al. [17] estenderam esse trabalho permitindo transferências de dados com maiores velocidades entre grupos, necessidade de menor número de reenvios (hops) para transferir mensagens entre grupos, menor número de dispositivos necessários por área geográfica, e uso de recursos mais equilibrado.

Bluetooth. O Bluetooth é uma tecnologia sem fios padronizada pelo *Bluetooth Special Interest Group (SIG)* para troca de dados entre dispositivos a curta distancia. Uma rede formada por dois dispositivos conectados por Bluetooth é designada de *piconet*.

As *piconets* são compostas por um líder (*master*), um máximo de sete dispositivos activos (*slaves*), e até 255 dispositivos inactivos. O master coordena as comunicações com os slaves. Os slaves não podem comunicar entre eles, apenas comunicam com o master. Os inactivos não podem comunicar até que mudem para um estado activo (master ou slave). O papel de cada dispositivo na piconet pode mudar, no entanto nunca pode existir mais que um master e sete slaves.

Usando Bluetooth é possível formar uma rede com múltiplos grupos, designando-se de *scatternet* (e.g., [20]), onde um master ou slave pode estar conectado a uma segunda piconet. Neste caso, o nó alterna a comunicação entre as duas piconets por intervalos de tempo.

2.3 Persistência dos Dados

Os ambientes móveis, como os que estamos a considerar, são altamente dinâmicos devido à mobilidade dos nós. Tanto a mobilidade dos nós, como as entradas e saídas de nós na rede, causam diversas alterações na topologia da rede.

Um sistema de armazenamento distribuído para redes móveis ad hoc tem de estar ciente da alta probabilidade dos nós abandonarem os sistema inesperadamente, causando perda de conteúdos. A solução para diminuir a perda de conteúdos devido às saídas de nós é criar múltiplas réplicas de cada conteúdo. Esta estratégia não é infalível, pois no limite, todas as localizações das réplicas de determinado conteúdo podem desconectar-se da rede ao mesmo tempo, levando à perda permanente desse conteúdo.

Para além de aumentar a garantia da persistência de dados, a replicação também pode aumentar o desempenho em sistemas distribuídos especialmente em redes móveis ad hoc. Existindo múltiplas réplicas de cada conteúdo, o nó que faz o pedido pode aceder à replica mais próxima da sua localização, reduzindo a latência para obtenção dos dados.

Métodos de Replicação. Métodos e estratégias de replicação para redes móveis ad hoc são um tema com uma grande quantidade de investigação. Existem diversas publicações com estratégias de replicação bastante diferentes umas das outras que têm como objectivo resolver problemas diferentes. Dado esta vasta gama de métodos de replicação, Derhab et al. [8] propõem uma classificação para os diferentes tipos de métodos de replicação. De acordo com os autores, existem dois tipos de métodos de replicação: protocolos adaptados a ambientes móveis, e protocolos não adaptados.

Os protocolos adaptados a ambientes móveis tentam resolver pelo menos um dos seguintes três desafios: partições da rede, escalabilidade, ou consumo de energia.

1. Protocolos conscientes da energia (e.g., [14]): Estes métodos detectam quando o nó está a ficar com pouca bateria e replica os dados antes que a bateria termine, tendo como objectivo reduzir o consumo de energia.
   A distribuição e o balanceamento da carga de trabalho dos nós é uma forma de reduzir o consumo de energia aos nós que contêm os dados mais populares.
2. Protocolos conscientes das partições da rede (e.g., [6]): Tentam prever se algum nó irá se desconectar da rede e replicar o seu conteúdo previamente, ou ou se poderá ocorrer alguma partição na rede e replicar os dados que estão unicamente disponíveis numa das possíveis futuras partições para a outra partição.
3. Protocolos escaláveis (e.g., [19]): Um protocolo escalável não tem um grande *overhead* conforme aumenta o número de nós na rede. Estes protocolos tendem a diminuir a quantidade de comunicações existentes entre os nós.

Os métodos de replicação podem tentar resolver mais que um destes três problemas, e o perfeito seria resolver os três problemas. Mas para o nosso

conhecimento e de acordo com Derhab et al., não existe ainda um protocolo que faça isso nem que seja simultaneamente escalável e consciente das partições.

Para além destas três características, um protocolo de replicação pode ser proactivo ou reactivo, dependendo se os dados são replicados periodicamente ou se o envio de réplicas é desencadeado devido a mudanças na topologia da rede.

Atualmente, ainda não abordámos protocolos adaptados a ambientes móveis. Temos um protocolo adaptativo à popularidade dos conteúdos, onde os conteúdos mais populares (pedidos mais vezes) são replicados por um maior número de nós de maneira a distribuir melhor a carga e tentar poupar a bateria dos dispositivos que replicam os dados mais populares.

2.4 Recursos Limitados

Apesar das múltiplas funcionalidades que os dispositivos móveis apresentam hoje em dia, outras continuam a ser limitativas. Os *smartphones* de hoje em dia têm processadores, memória principal e espaço em disco que rivalizam com os computadores pessoais de poucos anos atrás, mas a bateria continua a ser um grande entrave ao que se consegue fazer num dispositivo deste tamanho. Assim, o uso dessas capacidades deve ser cuidadosamente gerido de maneira a não esgotar a bateria rapidamente.

Pela simples razão de que a bateria é o recurso mais limitativo dos dispositivos móveis de hoje em dia, optámos por implementar um mecanismo com o objetivo de poupar energia quando o dispositivo se encontra num estado de bateria fraca. Este mecanismo será descrito em detalhe mais abaixo.

3 Uma Abordagem Prática

Nesta secção, apresentamos o desenho da nossa solução para um sistema de armazenamento e partilha de dados distribuído e totalmente descentralizado para redes de dispositivos móveis.

O nosso sistema foi desenvolvido como uma biblioteca e segue o modelo clássico de armazenamento com pares chave/valor, onde cada chave serve como identificador e é mapeada para um valor imutável. Atualmente, o sistema suporta as seguintes operações:

inserir$\langle chave, valor \rangle$ Armazena o par chave/valor dado no sistema, disponibilizando-o aos outros nós;

obter$\langle chave \rangle$ Procura no sistema pela chave especificada, retornando o valor associado caso exista;

remover$\langle chave \rangle$ Remove o conteúdo mapeado pela chave especificada, caso o conteúdo tenha sido colocado por este nó; e

pesquisar$\langle filtro \rangle$ Retorna todas as chaves que estão em conformidade com o filtro fornecido.

Uma vez que estamos a considerar um ambiente com dispositivos móveis, o sistema foi desenvolvido tendo em conta algumas propriedades desses ambientes, nomeadamente (i) a persistência dos dados; e (ii) a consciência de energia.

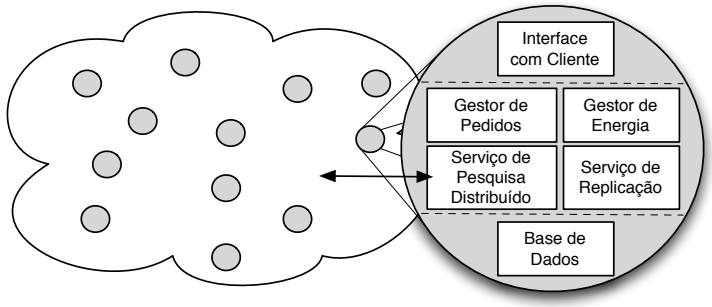


Figura 1. Arquitectura do sistema.

Para garantir a persistência dos dados no sistema é utilizada uma abordagem de *melhor esforço*. Dada a natureza volátil do ambiente que estamos a considerar, os efeitos do *churn* dificultam que o sistema garanta persistência "absoluta" dos dados, ou seja, estes permanecerão no sistema desde que todos os nós que armazenam um item não se desconectem do sistema no mesmo exato momento.

Uma vez que estamos a considerar dispositivos móveis, o sistema tem de empregar mecanismos de poupança de energia, caso contrário o seu uso torna-se pouco convidativo (ou até impeditivo). O nosso sistema tem consciência da energia do dispositivo, proporcionando um mecanismo para economizar energia (i. e., um modo de economia de bateria) quando o nível de energia de um dispositivo está abaixo de um limiar pré-definido. Ao entrar no modo de economia de bateria, o cliente ainda pode publicar e obter conteúdos, mas deixa de receber pedidos de outros nós. Mais informação sobre esta propriedade será dada mais abaixo.

3.1 Arquitetura do Sistema

Com o objetivo de implementar um sistema justo (em termos de recursos consumidos), optámos por uma arquitetura completamente descentralizada e simétrica, ou seja, todos os nós têm o mesmo papel no sistema e executam os mesmos serviços. Um nó é composto por vários componentes, apresentados na Figura 1, seguindo um modelo de três camadas e separando de forma clara as camadas de apresentação, lógica, e dados.

Interface com o Cliente. Fornece a interface para as aplicações cliente, fornecendo as operações disponíveis (`inserir`, `obter`, `remover`, e `pesquisar`) como uma biblioteca.

Gestor de Pedidos. O Gestor de Pedidos (GP) coordena o processamento dos pedidos enviados pelas operações realizadas na interface. As operações `obter` são primeiramente direcionadas para a camada de dados local (Base de Dados), para verificar se podem ser servidas localmente. Se necessário, só de seguida, é que são

transmitidas para o Serviço de Pesquisa Distribuído (SPD) para ser realizada uma pesquisa em todo o sistema. As operações `inserir` são tratadas pelo Serviço de Replicação (SR) que trata de gerir as réplicas necessárias nos nós remotos. As operações `pesquisar` são retransmitidas para a camada de dados local e para o SPD, para uma cobertura completa do espaço de pesquisa.

Serviço de Pesquisa Distribuído. A função do SPD é determinar a localização dos itens publicados por nós remotos, e executar todas as tarefas de manutenção relacionadas com a rede, tais como a gestão dos nós vizinhos. A implementação do nosso protótipo atual recorre ao TomP2P [4], uma Tabela de Dispersão Distribuída implementada em Java e que funciona em Android. A fim de tornar o TomP2P adequado a dispositivos móveis e redes móveis, foram realizadas várias otimizações, das quais destacamos a redução da troca de mensagens em vários casos: (i) o mecanismo de replicação original transmitia o conteúdo a ser replicado de forma pro-ativa infinitamente para todos os nós encarregues da replicação. Atualmente, é aplicada uma estratégia reativa, propagando unicamente os conteúdos a replicar quando são detetadas alterações no grupos de nós que replicam os dados; e (ii) as operações de pesquisa e de obtenção de dados, que antes faziam *flooding* na rede para determinar os nós que guardam os itens, utilizam agora uma estratégia de potência de duas escolhas [12], que só requer dois nós selecionados aleatoriamente. Caso estes dois nós não tenham o conteúdo desejado, serão selecionados outros dois nós e assim sucessivamente até o conteúdo ser encontrado. Esta abordagem é susceptível de aumentar a latência das operações, mas reduz o número de mensagens trocadas e, por conseguinte, a energia consumida.

Serviço de Replicação. A replicação de dados é uma técnica conhecida para tentar garantir a persistência de dados em sistemas distribuídos. Apropriadamente, o SR é responsável pela manutenção do número de réplicas para cada item publicado de acordo com uma estratégia de replicação predefinida. O sistema cresce a partir do TomP2P e é completamente descentralizado, distribuindo a responsabilidade de garantir o nível de replicação de dados de cada item por vários nós. Assim, para cada item, há um nó *responsável* por gerir o número designado de réplicas, detetando eventuais desconexões de nós que contenham réplicas. Para cada desconexão, o nó responsável seleciona um novo nó e envia uma réplica para esse mesmo nó. A desconexão de um nó responsável desencadeia a eleição de um novo nó responsável.

Atualmente, o sistema suporta três estratégias de replicação diferentes: (i) *replicação estática*; e (ii) *replicação com reconhecimento da rede* estão incluídas no TomP2P; (iii) *replicação por popularidade* regista o número de pedidos de cada item e aumenta o número de réplicas para os itens com mais acessos. Esperamos que este mecanismo reduza a latência na obtenção de um item popular e ao mesmo tempo reduza o consumo de energia nos nós que armazenam estas réplicas, simplesmente porque os dados populares serão difundidos entre um maior número

de nós, conseguindo assim um melhor balanceamento de carga ao solicitar esses itens.

Gestor de Energia. Quando o nível da bateria fica abaixo de um limiar pré-definido o nó é parcialmente desativado. O Gestor de Energia (GE) desliga as tarefas de manutenção relacionadas com a rede no SPD, fazendo com que este nó fique *invisível* para os restantes nós do sistema. Além disso, o GE desliga o SR para terminar as tarefas de replicação. Esta abordagem reduz o consumo de energia do nó, e o sistema encontra novas localizações para guardar réplicas existentes no nó parcialmente desativado. Se o nó for recarregado e o nível da bateria ultrapassar o limite, o nó volta a juntar-se totalmente ao sistema e continua a funcionar como antes.

Base de Dados. Uma base de dados do tipo chave/valor para mapear as chaves para os conteúdos partilhados e obtidos.

4 Resultados Experimentais

Nesta secção mostramos e analizamos alguns resultados experimentais do nosso protótipo. O nosso sistema foi desenhado como uma biblioteca, e com essa biblioteca desenvolvemos uma aplicação Android para partilha de fotografias. Assim, temos dois níveis de avaliação: (i) avaliação do protótipo Android, que permite aferir acerca da usabilidade do nosso sistema ao nível do utilizador; e (ii) avaliação da biblioteca num computador, que permite uma avaliação com um número maior de nós.

4.1 Protótipo em Android

Desenvolvemos uma aplicação móvel em Android para a gestão de uma galeria de fotografias. A aplicação oferece várias *vistas*, nomeadamente para a visualização de conteúdos privados (locais), de conteúdos partilhados pelo próprio utilizador, de conteúdos que foram obtidos de dispositivos remotos e uma lista de nomes e *thumbnails* dos conteúdos disponíveis no sistema. Para os conteúdos privados, o utilizador tem a possibilidade de escolher e partilhar o que desejar. A partilha de uma fotografia faz o sistema gerar um identificador exclusivo que é armazenado na BD e publicado no sistema. Os conteúdos partilhados podem ser removidos do sistema apenas pelo utilizador que partilhou o conteúdo. No entanto, caso o conteúdo tenha sido obtido por outro dispositivo não é possível removê-lo do dispositivo remoto, ficando apenas indisponível para futuras obtenções.

Para avaliar o impacto da nossa implementação no consumo de bateria de um *smartphone*, realizámos dois testes preliminares numa rede composta por dois tipos de dispositivos: *tablets* Nexus 7 de 2013 com Android 5.1 e *smartphones* Motorola Moto G de 2ª geração com Android 5.0.2. O primeiro teste teve como objetivo determinar o consumo de bateria quando nenhuma operação é realizada

pelo sistema, para além da manutenção da rede pelo SPD. Primeiro, quisemos determinar o consumo de bateria para a criação de uma rede através do modo de *ponto de acesso* de um dispositivo. Para isso, criámos uma rede onde o Nexus 7 tinha o papel de *ponto de acesso* e os Motos G juntaram-se à rede. Numa execução de 8 horas, o consumo de bateria foi de 24% e 6% para o dispositivo que criou a rede (ou seja, o ponto de acesso) e os dispositivos que se juntaram, respetivamente.

Numa execução com a mesma duração (8 horas), mas agora com a nossa aplicação a correr, observámos um aumento no consumo de bateria de 11% e 2%, em relação à execução anterior, para o dispositivo que criou a rede e os dispositivos que se juntaram, respetivamente.

Estes valores foram obtidos através do cálculo da média de duas observações, onde se verificaram variações de apenas 2% e 4%, para o Moto G e Nexus 7, respectivamente. Não foram realizados testes em que o ponto de acesso é oferecido pelo Moto G por falta de suporte de tal funcionalidade nesses dispositivos. No entanto, não é necessário que seja um dispositivo a fornecer a rede, podendo esta ser formada a partir de um *router* disponível na área. Dessa forma, evita-se gastos de bateria na criação do *ponto de acesso* por parte de um dispositivo.

Num segundo teste, pretendemos determinar se o envio de pedidos e obtenção de conteúdos consumia muita bateria. Neste teste, o Nexus 7 estava continuamente a fazer pedidos e a obter (e apagar, devido a limitações de armazenamento no dispositivo) uma imagem de 2 MB durante uma hora e o Moto G a responder aos pedidos, enviando a imagem. No final, o Nexus 7 obteve e armazenou em disco a imagem 4962 vezes usando 10% de bateria. De notar que também estava a servir de ponto de acesso. Por sua vez, o nó que serviu os pedidos (Moto G) teve um consumo de bateria de 14%.

4.2 Avaliação

Em seguida apresentamos alguns testes que demonstram os resultados das otimizações que realizámos ao TomP2P. Os testes foram executados numa máquina, onde foram criadas várias instâncias de nós (um processo para cada nó). A máquina onde os testes foram executados tem um processador Intel Core i5-2430M 2.4GHz, 4 GB de memória RAM e corre o sistema operativo Linux, distribuição Ubuntu 14.04.1.

Se nada for dito em contrário, todos os testes foram executados com as seguintes configurações fixas: (i) o tempo de execução dos testes foi de 20 minutos; (ii) o período de intervalo de execução das tarefas de manutenção da replicação foi de 60 segundos; e (iii) o número mínimo de réplicas que o sistema manteve para cada item foi de 5 réplicas.

Otimização do mecanismo de replicação. Um bom mecanismo de replicação que não cause um *overhead* exagerado, mas que consiga manter os dados no sistema é importante para qualquer tipo de sistema distribuído que necessite de manter os dados disponíveis. Dada as características dos ambientes móveis, este componente é ainda mais importante.

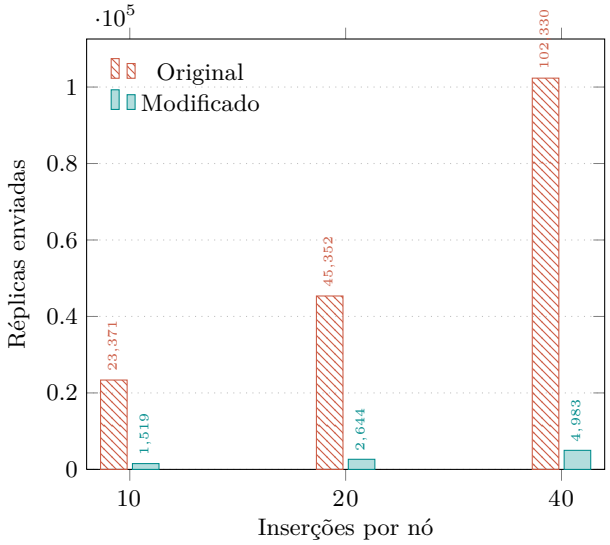


Figura 2. Número de réplicas enviadas por inserções.

Para adaptar o mecanismo de replicação existente no TomP2P, fizemos algumas modificações de forma a reduzir o número de réplicas transmitidas. O protocolo original tem um mecanismo proativo, ou seja, periodicamente envia as réplicas para os seus destinos de replicação mesmo que o destinatário já tenha os dados. Esta característica poderá não ser um grande problema em redes convencionais (como a Internet), dependendo da quantidade e tamanho dos conteúdos, velocidade da rede, número de nós, etc. No entanto o mesmo não se verifica em rede móveis, devido à largura de banda e bateria limitadas.

Dado este problema, alterámos o protocolo para um mecanismo reativo, onde as réplicas são enviadas apenas quando (e se necessário) há alterações na rede.

Nos gráficos das Figuras 2 e 3 podemos observar a quantidade de réplicas enviadas pelo protocolo original (*Original*) e pela nossa modificação (*Modificado*). Para os resultados do gráfico da Figura 2 foram utilizados 20 nós. Para os resultados do gráfico da Figura 3 foram feitas 10 inserções de conteúdos por cada nó.

Nestes gráficos podemos verificar que conseguimos reduzir bastante as réplicas transmitidas. Neste caso, o sistema esteve em execução apenas durante 20 minutos, numa execução com um período de tempo mais alargado estas diferenças seriam ainda mais acentuadas.

Número de réplicas transmitidas com *churn*. O *churn* é um problema com um impacto maior em ambientes móveis. Dado este problema, queremos avaliar o sistema quando está exposto a níveis de *churn* diferentes.

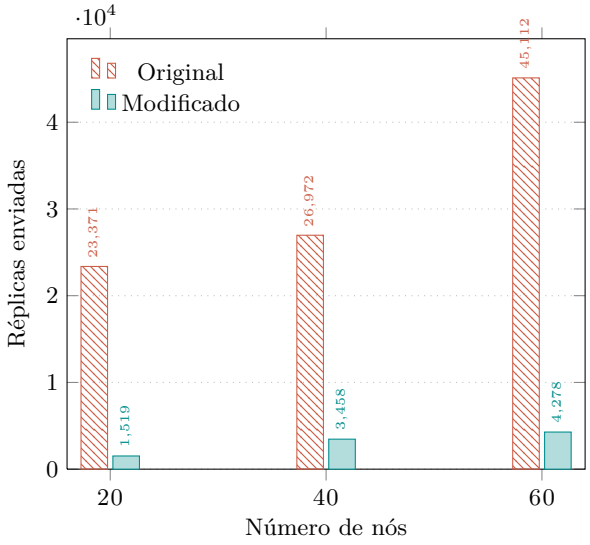


Figura 3. Número de réplicas enviadas por número de peers.

No gráfico da Figura 4 podemos visualizar os resultados de uma execução com 20 nós, onde cada um fez 10 inserções no sistema. Os valores de *churn* no gráfico denotam o número de nós que se desconectaram do sistema em intervalos de 60 segundos. Após a desconexão dos nós, novos nós (em igual número) foram conectados ao sistema.

Como era esperado, os resultados do protocolo de replicação original são praticamente iguais aos resultados sem *churn*, pois este protocolo funciona de forma proativa, enviando réplicas periodicamente independentemente se há ou não alterações na rede.

No protocolo de replicação modificado, podemos verificar que o número de réplicas aumenta bastante. No entanto, como estamos a desconectar 25% e 50% dos nós existentes (5 e 10 nós, respetivamente) é necessário repor uma grande quantidade de réplicas nos novos nós. Tendo em conta que são 200 conteúdos e 15 nós (depois da desconexão de 5 nós) para enviar réplicas aos novos 5 nós, por minuto há uma média de 230 réplicas enviadas, o que dá cerca de 15 réplicas enviadas por nó por minuto. Nos resultados obtidos na avaliação do protótipo em Android, onde um dispositivo fez quase 5000 pedidos, guardou e apagou o ficheiro do disco, obtendo uma média de 83 ficheiros pedidos por minuto. Cruzando os dois resultados, podemos concluir que o envio de 15 réplicas por minuto, por nó, não é muito significativo.

5 Trabalho Relacionado

Nesta secção apresentamos projectos relacionados com armazenamento distribuído em redes ad hoc, e com comunicações ad hoc para dispositivos móveis.

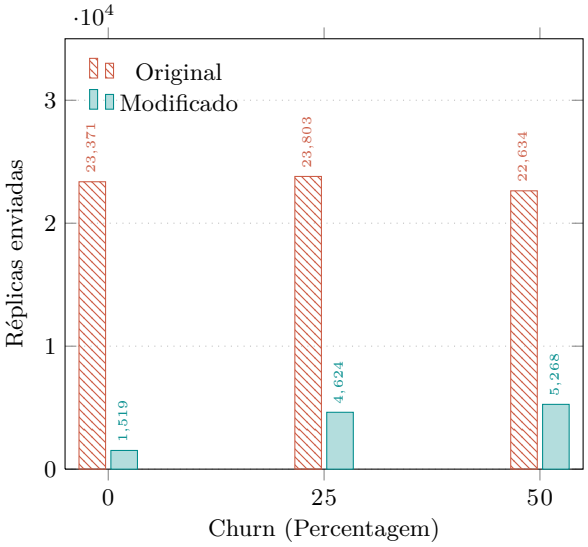


Figura 4. Número de réplicas enviadas consoante o nível de *churn*.

Phoenix [13] é o sistema que, para nosso conhecimento, tem os objectivos mais semelhantes ao que pretendemos: garantir persistência e disponibilidade de dados. Os autores propõem um protocolo baseado em rondas para a manutenção das réplicas no sistema através de transmissões *broadcast* e *single-hop*. Em cada ronda, é elegido um bloco (os conteúdos são partidos em diversos blocos) para verificação do número de réplicas existentes e, se necessário, enviar o bloco a nós que não o tenham para repor o número mínimo de réplicas existentes. No entanto, não é proposto nenhum mecanismo de pesquisa e obtenção de dados existentes no sistema.

O projecto iTrust [11] permite publicar, pesquisar e obter conteúdos através de uma rede móvel ad hoc constituída por dispositivos Android com WiFi Direct. Neste sistema o dispositivo que quer partilhar conteúdos envia várias cópias de meta-dados para múltiplos dispositivos remotos aleatoriamente. Estes meta-dados contêm a localização do conteúdo. Para obter conteúdos, o dispositivo envia pedidos para dispositivos remotos aleatoriamente até encontrar algum que contenha os meta-dados. De seguida, faz o pedido ao dispositivo que contem o conteúdo. Uma limitação da abordagem utilizada neste sistema é a impossibilidade de garantir persistência de dados. Caso o dispositivo que partilha determinado conteúdo se desconecte da rede, o conteúdo perde-se.

Luo et al. [10] propõem um sistema de armazenamento baseado em quóruns probabilísticos, permitindo acessos de leitura e escrita concorrentes. Os pedidos de leitura e escrita são difundidos até atingirem o número de nós necessários para ter o quórum de leitura ou escrita, respectivamente. Esta característica causa a necessidade de configuração para diferentes ambientes (número de nós, área, taxa de leituras e escritas, etc.). Este sistema não é totalmente descentralizado

e simétrico, ou seja, a tarefa de armazenamento de dados não é partilhada por todos os nós, havendo a eleição de alguns servidores (onde serão guardados os dados) na inicialização do sistema. A avaliação e testes foram realizados em um ambiente de simulação, não havendo uma implementação funcional para dispositivos.

Thomas e Robble desenvolveram o projecto Smart Phone Ad-Hoc Networks (SPAN) [18] e têm o objectivo de criar uma rede ad hoc de dispositivos Android. Para comunicar através de WiFi em modo peer-to-peer os dispositivos têm de ser reconfigurados, o que requer fazer desbloquear o dispositivo (fazer *root*) para obter permissões de super-utilizador. Esta necessidade causa dificuldades na sua utilização por parte de utilizadores comuns. Para além disso, também funciona apenas numa pequena gama de modelos existentes.

6 Conclusão

Neste artigo apresentamos um sistema de armazenamento descentralizado para redes de dispositivos móveis. O sistema foi desenvolvido como uma biblioteca e desenvolvemos uma aplicação móvel para Android usando essa biblioteca. Os resultados experimentais retirados com os dispositivos móveis mostram que o nosso sistema pode ser usado continuamente durante um evento, como uma festa, um concerto ou um evento desportivo, para partilhar conteúdos sem esgotar a bateria do dispositivo. Os testes de maior escala mostram indícios de que as otimizações realizadas também apresentam potencial para melhorar a eficiência e a escalabilidade do sistema.

Como trabalho futuro temos ainda a realização de uma avaliação mais extensa e com um maior número de máquinas, de maneira a perceber melhor qual o impacto das otimimizações realizadas: os prós e contras do protocolo de replicação por popularidade para os dados mais populares e menos populares, e se diferentes níveis de *churn* também afetam o *overhead* e capacidade de persistência de dados deste protocolo; número de mensagens de pedidos necessárias, e tempo que demora, para obtenção de conteúdos com os diferentes protocolos de replicação; tempo para obtenção de dados para conteúdos de diferentes tamanhos; e, capacidade de manter os conteúdos nos sistema para os diferentes protocolos de replicação;

Atualmente, o protótipo em Android necessita de um ponto de acesso para a comunicação entre os vários dispositivos, como um *router* ou um *ponto de acesso* móvel. O nosso próximo passo é permitir comunicações ad hoc (por exemplo, utilizando comunicações entre múltiplos grupos WiFi Direct) e, com isso, extender a aplicabilidade do nosso sistema para cenários onde não existem pontos de acesso sem fios ou até mesmo quando a rede está sobrecarregada.

Referências

1. Bluetooh, Bluetooth Special Interest Group (SIG). `http://www.bluetooth.com`
2. The Serval Project. `http://www.servalproject.org`, acedido a: 2015-07-14
3. WiFi Alliance, WiFi Direct. `http://www.wi-fi.org/discover-wi-fi/wi-fi-direct`

4. Bocek, T.: TomP2P, A P2P-based high performance key-value pair storage library. http://www.tomp2p.net/
5. Casetti, C., Chiasserini, C., Pelle, L.C., Valle, C.D., Duan, Y., Giaccone, P.: Content-centric routing in wi-fi direct multi-group networks. In: WoWMoM (2015)
6. Chen, K., Nahrstedt, K.: Integrated data lookup and replication scheme in mobile ad hoc networks. In: ITCom. pp. 1–8 (2001)
7. DARPA Strategic Technology Office: Content-based mobile edge networking (CBMEN). http://www.darpa.mil/program/content-based-mobile-edge-networking, acedido a: 2015-07-14
8. Derhab, A., Badache, N.: Data replication protocols for mobile ad-hoc networks: a survey and taxonomy. IEEE Communications Surveys and Tutorials 11(2), 33–51 (2009)
9. Drolia, U., Martins, R., Tan, J., Chheda, A., Sanghavi, M., Gandhi, R., Narasimhan, P.: The case for mobile edge-clouds. In: UIC/ATC '13. pp. 209–215. IEEE Computer Society (2013)
10. Luo, J., Hubaux, J.P., Eugster, P.T.: Pan: Providing reliable storage in mobile ad hoc networks with probabilistic quorum systems. In: MobiHoc. pp. 1–12. ACM (2003)
11. Michel Lombera, I., Moser, L., Melliar-Smith, P., Chuang, Y.T.: Mobile ad-hoc search and retrieval in the itrust over wi-fi direct network. In: ICWMC. pp. 251–258 (2013)
12. Mitzenmacher, M.: The power of two choices in randomized load balancing. In: IEEE TPDS (2001)
13. Panta, R.K., Jana, R., Cheng, F.T., Chen, Y.F.R., Vaishampayan, V., et al.: Phoenix: Storage using an autonomous mobile infrastructure. IEEE TPDS 24(9), 1863–1873 (2013)
14. Shinohara, M., Hayashi, H., Hara, T., Nishio, S.: Replica allocation considering power consumption in mobile ad hoc networks. In: IEEE PerCom Workshops). pp. 463–467 (2006)
15. Stoica, I., Morris, R., Karger, D.R., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM. pp. 149–160 (2001)
16. T. Klinberg and R. Manfredi: Gnutella protocol specification v0.6. http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html (June 2002), acedido a: 2015-07-14
17. Teófilo, A., Remédios, D.: Group-to-group bidirectional Wi-Fi Direct communication with two relay nodes. In: MobiQuitous (2015), to appear
18. Thomas, J., Robble, J., Modly, N.: Off grid communications with android meshing the mobile world. In: IEEE-HST. pp. 401–405 (Nov 2012)
19. Yu, H., Martin, P., Hassanein, H.: Cluster-based replication for large-scale mobile ad-hoc networks. In: WIRELESSCOM. vol. 1, pp. 552–557. IEEE (2005)
20. Zaruba, G.V., Basagni, S., Chlamtac, I.: Bluetrees-scatternet formation to enable bluetooth-based ad hoc networks. In: IEEE ICC. vol. 1, pp. 273–277. IEEE (2001)

SecureTracking

Mobile Tracking for Children and Mentally Ill Patients*

Rui Morais, Ricardo Chaves, and João Barreto

INESC-ID / Instituto Superior Técnico, Universidade de Lisboa

Abstract. Currently there are several systems dedicated to monitor the position of objects, vehicles and people. Most of these monitoring systems focus on the energy efficiency, implementing strategies to maximize battery lifetime by carefully choosing the localization method, or using alternative sensors like accelerometer and compass to estimate the current location. When monitoring the position of a person, especially children or people with mental disabilities, security and privacy issues arise. However, most of the existing tracking systems do not take into account the security issue when transmitting the obtained information and the localization process itself. To fulfil these requirements, this paper presents the proposed SecureTracking system, a tracking system capable of monitoring the position of a person, applying some control on that position through geofencing, while ensuring the confidentiality, integrity, authentication and freshness of the transmitted information. The proposed solution also considers anti-spoofing mechanisms on the obtained location information. A fully functional prototype was implemented using Android smartphones and a dedicated server, acting as an intermediate, possibly insecure, storage point. The system herein proposed shows that a low cost, energy efficient, and secure tracking solution can be deployed using off-the-shelf mobile devices.

1 Introduction

Nowadays, tracking systems are becoming increasingly important in our daily life. Many people use tracking services to monitor the location of mail packages, company vehicles or simply as a recreational way of keeping track of friends or relatives, for example, in social networks. Voluntary sharing the current location of objects and persons has become ordinary, being eased by the growing use of smartphones and other embedded devices (e.g., digital cameras, vehicular on-board computers, etc.) equipped with location sensors and mechanisms capable of obtaining their position.

* This work was supported by project EMC2 (co-financed the ARTEMIS Joint Undertaking under grant agreement no. 621429), "TRACE" (co-financed by the European Commission through the contract no. 635266) and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

Such use of the locating capabilities usually does not require dedicated security measures since the information is saved and shared voluntarily. But, on the other hand, if a tracking system is meant to maintain someone safe, the security requirements are much higher and many risks must be taken into account. In this paper, the main focus is the tracking of children and mentally ill persons. Spoofing attacks may lead to incorrect localization, attacks on the communication channels may reveal the location to unauthorized individuals, or allow tampering with the exchanged messages, pretending normal operation. A secure tracking system must aim, not only at obtaining the person's correct position, but also at ensuring the confidentiality, integrity, authenticity and freshness of that information to ensure the person's own safety. These security issues become even more crucial when monitoring the position of persons belonging to a critical group, such as children and people with mental disabilities.

There is a trade-off between security measures and acceptable privacy loss when monitoring the position of a person. Reporting a more accurate location could be better help in dangerous situations (such as kidnapping), but, on the other hand, every day use does not require such accuracy and it can be seen as being too much intrusive. Tracking systems that monitor the position of humans should not abuse in terms of privacy invasion and their main focus, when not recreational, should be keeping them safe. There is another trade-off between security and efficiency, since the former leads to increased resource consumption, but the latter is required to maximize the lifetime of the system. To the best of our knowledge, the tracking solutions which design and implementation are known sacrificed security for the benefit of energy efficiency.

This paper proposes the SecureTracking, a tracking solution capable of obtaining a person's position and communicating that information, or any other relevant, to a tutor. It also supports some control on that location through geofencing and allows the tutor to ask and get the monitored person's position on-demand. While providing these features, the proposed system ensures confidentiality, integrity, authenticity and freshness of the exchanged messages, while at the same time considering efficiency as an important factor to maximize battery lifetime. It should also be easy and comfortable to transport by the monitored person and takes into account privacy and ethical issues as far as possible.

One of the main contributions of this paper is the system design, since it allows many tutors to monitor the same person while ensuring the security requirements. The experimental evaluation shows the feasibility of SecureTracking as a secure, low cost and energy efficient monitoring system which can be deployed using off-the-shelf mobile devices.

Section 2 details some concepts in the context of the related work and this paper, and an overview on some of the current tracking solutions. Section 3 describes the proposed solution taking into account the defined requirements and the analysis of the state-of-the-art. The Proposed Solution is followed by the Implementation section, which details the proposed solution targeting an Android system. Section 5 explains how the implementation was characterized

and evaluates it. Section 6 concludes this paper presenting an overall analysis of the proposed system and some remarks on possible future work.

2 State of the art

There are various uses for tracking systems, from monitoring mail packages and vehicles to tracking children on their path to school or dementia patients. Different uses also imply different functionalities, characteristics, concepts and methods of localization.

To have some control on the location of a children or a dementia patient, it is useful to apply the concept of Geofencing. The Geofencing technique consists on the combination of two operations: first, regular monitoring of the position of some mobile device; and on the other hand the definition of a geographic area of interest, called geofence (geographic fence). The whole process consists on verifying if the position of the mobile device is inside or outside that area. Geofencing can be performed using different techniques: using a polygon where the vertices are defined by coordinates; using a circular area defined by the coordinates of its center and a radius; or verifying if the monitored device follows or keeps close to a particular route.

2.1 Localization Methods

There are various methods already studied and put in practice to obtain a location using mobile devices like smartphones. These methods differ from each other in characteristics such as accuracy, speed, processing cost and resource and energy consumption. Among the most commonly used methods nowadays, using mobile network antennas (e.g. GSM antennas), using Wi-Fi Access Points (APs) and A-GPS are perhaps the most popular.

Mobile Network Antennas

Cell-ID The localization process using mobile antennas is made using some parameters, namely the cell identification (Cell-ID) and the Location Area Identifier (LAI) [1]. A Base Transceiver Station (BTS) covers a certain area which is composed by various cells, each of them with a unique identifier. The cells are grouped in greater groups identified by the LAI. A mobile device always knows its Cell-ID (the cell where it is) and the LAI, and from those values it can estimate its location. This estimate depends on the density of antennas nearby, interference that results in signal noise and the choice of the antenna to connect (as it may not be the closest one [1]).

Tests conducted in [2] determined an accuracy of about 500 meters in urban environment and accuracy in a range of 1 to 10 kilometers in sub-urban environment when locating using Cell-ID.

Lateration and Angulation Besides the localization using Cell-ID and LAI, it is also possible to obtain the location by triangulation of mobile antennas through lateration and angulation.

Lateration consists in determining the position using the distance to different points. For example, a mobile device can calculate its location using the approximate distance to the mobile antennas. In a bi-dimensional plan, three reference points and the approximate distance to each of them are required to determine the position. Angulation consists in using angles to determine a position. For example, being possible to know the direction of the signal of two antennas and knowing the distance between them, it is possible to determine the current position. However, this solution needs to know the distance between each antenna.

Wi-Fi APs Localization using Wi-Fi Access Points (APs) signals, also known as Wi-Fi-based Positioning System (WPS), follows a similar pattern as Cell-ID positioning: analysing nearby Wi-Fi APs regarding their identifiers (Media Access Control and Service Set Identifier [3]) and knowing their approximate location allows a device to locate itself. The location and identifiers of the Wi-Fi APs needs to be collected and registered in databases so that they can be used by nearby devices to locate themselves. The operation of registering the location and identifiers is known as "wardriving" and has been carried out by different entities [4,5]. When a device tries to obtain a location, it uses the identifiers of the nearby Wi-Fi APs and queries a database where this information is mapped together with the respective locations of those APs. The accuracy of this method is influenced by the number of Wi-Fi APs nearby and the fact that those may have or may have not been mapped already. Tests conducted by [6] revealed a mean accuracy of approximately 30 meters.

A-GPS The GPS is composed by a constellation of 27 satellites (24 active and 3 in standby). The navigation system provides information about the position of each satellite and precise time set by the satellite atomic clocks. Each satellite is uniquely identified by a code that is also broadcast. To obtain its location, a GPS receptor calculates the distance to various satellites at the same time, given that a minimum of 3 satellites are required to obtain a location. Fixing at satellites may take some time, known as the Time-To-First-Fix (TTFF), factor that has a great impact on this method performance, speed and time/energy consumption. However, this can be improved using Assisted-GPS that uses a secondary source from which it downloads the almanacs and ephemerides, also known as assisting data or Xtras, that contain information on the current state of the satellite constellation, precise data on time and location and orbit data of satellites. Tests conducted in [6] revealed a mean accuracy of 10 meters using GPS, and according to [2] the accuracy may vary between 10 to 100 meters in urban and 10 to 20 meters in suburban environment.

Security Analysis There are security issues that must be considered when interacting with the infrastructures of the GSM network, the Wi-Fi APs and A-GPS.

The original architecture of the GSM network intended to authenticate its mobile users, maintain confidentiality of users' data and signal information, the anonymity of theirs subscribers and the usage of the SIM card (Subscriber Identity Module) as a security module. However, the authors of [7,8,9] point out flaws in the existing structure, namely that the A5 algorithm, which is used in encryption, is considered weak and susceptible to cryptanalysis, the network does not provide end-to-end security, and also it does not provide mechanisms for mutual authentication. The authors of [7] propose establishing a secure channel at the end-entities of the communication, such as the use of ciphering keys.

In addition to being susceptible to spoofing and jamming attacks, WPS can also suffer injection or corruption attacks on the databases that hold the mapping between Wi-Fi APs and their locations [10]. The same authors propose sanity tests at the client through historical registry of APs to detect unusual discrepancies and higher security on the databases side to avoid tampering with the data.

According to [11], A-GPS is also vulnerable to jamming and spoofing attacks. The same author proposes some countermeasures to prevent the latter attacks. For example, it is possible to analyse the GPS signal strength and detect if it is too strong compared to a certain threshold or if it suffers sudden changes that could indicate a fake signal being inserted. Another possible technique is monitoring the number and identification of GPS satellites signals since GPS satellite simulators usually broadcast a greater number of signals against what is usually detected. Dead reckoning can also be used as a sanity test by comparing its data with the locations obtained using A-GPS.

2.2 Related Work

There are many systems that aim at monitoring the position of persons. In this section, some of these solutions are analysed considering their main features and the context and requirements defined in this paper.

Most of the current smartphones have location services and sensors, such as A-GPS, Wi-Fi and Cell-ID, and mobile applications are able to make use of them to obtain the current location. Applications like "Life360 - GPS Tracking" [12] and "Real Time GPS Tracker" [13] provide ways to share the location with a group of people set by the user and receive location information from other people. These applications propose a system like a social network. There are no concerns about security issues because the responsibility lies on the user when installing the application, granting it permission to access the smartphone sensors, and it is the users will to share their location with each other. Another type of application is the "MobileMonitor" [14] which works together with a private web portal and allows uploading to the system servers detailed information about the device's state, like phone call and messaging history, last position, files and operating system data. This system collects too much information from the

monitored device. Although the communication is ciphered, there is no information about cipher key exchanges processes, and there are no mechanisms on anti-spoofing.

Some other solutions like the one in [15] analyse the behaviour of an accelerometer to determine the transportation mode in order to dynamically select the location provider and the interval of the location requests. Using this technique allows reducing the need to use GPS in about 96% and increase battery lifetime by about 75%. Another example is WheelLoc [16] that combines the accelerometer and magnetometer together with a local Cell-ID location database and street map to estimate turns taken and distance travelled. This system matches these estimations with the map using a Hidden Markov Model (HMM) and Viterbi algorithm (VA), obtaining the location within 40 milliseconds with an accuracy of about 40 meters and low power consumption. Enhanced Localization Solution (ELS) [17] is another system than combines standard localization techniques in mobile phones (GSM, Wi-Fi, GPS) with data from other sensors (accelerometer, audio, compass, gyroscope) using Human Mobility modelling and machine learning techniques.

There are other solutions that apply dead reckoning to track the position. It is the example of UPTIME (Ubiquitous Pedestrian Tracking usIng Mobile phonEs)[18], which makes use of the accelerometer to estimate step counting and gait-based algorithm to detect the step size. This system achieved an average error of 6.9% in outdoor and 5.9% in indoor scenarios when tracking the position.

In [19], the authors propose a tracking system to be used on a mobile phone with GPS with some security mechanisms. The solution includes the development of a Java application to be installed on a mobile phone and a web portal to check the location and apply geofencing. The application uses the GPS of the mobile phone to obtain its location with a custom period set by the user himself. When encryption is activated, it uses the device's International Mobile Equipment Identity (IMEI) code to set a 128 bit encryption key, ciphers the GPS coordinates, using the Advanced Encryption Standard (AES) algorithm, and sends this information to a server to be inserted into a database.

In general, there are many solutions to monitor someone's position. Most of them focus on using the available resources efficiently, by dynamically choosing the adequate localization method, reducing the need to use GPS by combining other localization sensors like accelerometer and compass. Among the analysed solutions, only MobileMonitor [14] and [19] consider security measures, but they are limited to ciphering the exchanged messages. This is not satisfactory as they lack integrity, authenticity and freshness measures as well as anti-spoofing mechanisms.

3 Proposed Solution

As discussed in the previous section, although there are many tracking systems currently available, none of them fully meet the needed requirements. Considering the previously analysed solutions and the defined requirements, the Se-

cureTracking system is herein proposed. This system deploys a tracking solution that provides geofencing capabilities while providing security to ensure secure communication channels and anti-spoofing measures on the location process. The proposed system is suitable for scenarios where parents are concerned with kidnapping risks of their children, for example, in environments with much confusion where children may walk or travel without adult supervision. A similar scenario is related with dementia patients or people with mental disabilities. In these cases, usually, there is a responsible person, or more, assigned to them.

This system assumes the existence of two major components that play the main roles of a tracking system: the Monitored device and the Tutor device. The Monitored device needs to be transported by the person whose position is to be monitored. The system must have built-in modules capable of obtaining its current location, process and validate that acquired information, and send that information to the desired destination, eventually with some degree of interaction with the receiver. On the other hand, the Tutor device should be carried by the person, or persons, that control the position. There can be multiple Tutor devices, and the Monitored device is able to report to them. The Monitored device operates in two modes: normal mode, where it periodically reports its position, and safe mode, where it only sends a "heartbeat" message acknowledging that the device is working properly. The safe mode can be triggered within periods of time when it is expected that the device is secure. These periods of time are defined as SafeSchedules that are established by the tutor.

The simplified architecture is presented in Figure 1. The **M** block represents the Monitored side and the **T** block the Tutor side. The Monitored device gets its position using Wi-Fi APs, A-GPS or the mobile antennas, in that order of preference. The communication is assumed to be bidirectional between the Tutor and the Monitored devices. This approach also considers the presence of the repository server that acts as an intermediate in the Tutor-Monitored communication, storing messages and sending them to the appropriate receivers.

Any communication between the monitored device and the tutor is ciphered and authenticated. The keys to cipher and authenticate messages are exchanged during the initial setup, but can be renewed at the request of the tutor device. To assure freshness of the messages and avoid replay attacks, each message also contains a timestamp. To be valid, a message must have a timestamp greater than the last message received. The message format is detailed in Figure 2. The + signal represents a concatenation. Every exchanged message is ciphered with a secret key K_{enc}. The ciphered content is composed by the concatenation of the actual data MSG with a timestamp TS generated at that time, plus the MAC value of $MSG + TS$.

When a new position is detected, the system makes a simple comparison with the last position using the distance between them and the time since the last update. If the velocity is above a threshold, a message is sent to the tutor so he can investigate if it is a dangerous situation (a spoofing attack, a kidnap, or maybe just accuracy error). The keys for these security services are generated

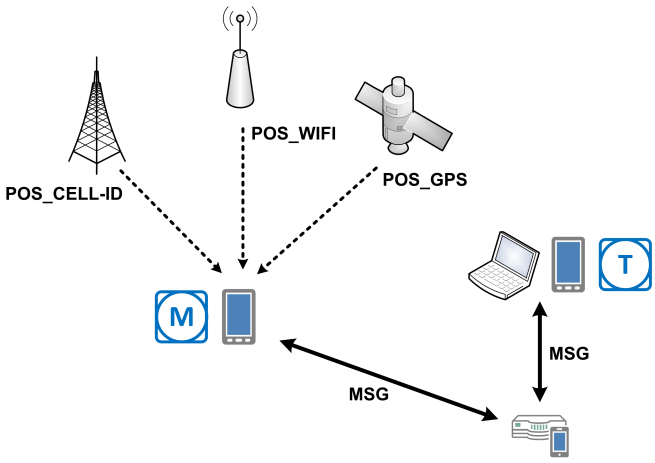


Fig. 1. Proposed Solution - SecureTracking system overview

and exchange at first setup, but the tutor is responsible for regenerating them every pre-defined time interval in order to avoid or minimize attacks. To do so, the tutor application supports key regeneration. To send the keys to the monitored device, the tutor uses a Key Encryption Key (KEK) dedicated to cipher the new key values.

The repository server is a simple backup system to be used as a database for the monitored device's messages. It does not hold any keys, so it cannot decipher nor validate received messages. To perform its role, the repository needs to be able to communicate via text-based SMS or messages through the Internet, so it has a GSM module and Internet connection.

4 Implementation

To build a prototype system, Monitored and Tutor devices were implemented on Android smartphones. As already stated, smartphones' usage has grown in the past few years, and most of them have Wi-Fi and A-GPS modules, besides GSM and 3G communication. Also, the APIs for smartphone development, namely Android, are usually well documented and provide access to the device's sensors, making them a viable solution. The Tutor's PC application and the Repository Server were implemented in Java.

4.1 Mobile Devices

The Monitored and Tutor devices make use of the Android Broadcast Receiver to register and detect certain events, called intents, such as receiving SMS, location requests or new location information. These intents are handled by a service that runs on the background. The Android Alarm Manager is also used to trigger the

K_{enc}	Secret key for cipher algorithm
K_{mac}	Secret key for mac algorithm
MSG	Data to send (e.g. location)
TS	Timestamp
$MSG + TS$	Concatenation of MSG with TS
MAC	HMAC-SHA256(K_{mac}, $MSG + TS$)
M	Final message to be sent

$$M = \{MSG + TS + MAC\}K_{enc}$$

Fig. 2. Message format

periodic location update in the Monitored device. The localization manager, in the Monitored device, is responsible for the process of obtaining its current position using the `LocationManager` of the Android API and its notion of `Location`. According to the Android API Reference, all locations obtained using this API include valid latitude, longitude, timestamp and accuracy information.

4.2 Repository Server

The Repository Server prototype was implemented using a smartphone with GSM, connected to a personal computer, and internet connection. Using the smartphone GSM communications it was possible to simulate a server with a GSM module, capable of receiving and sending SMS messages from and to the mobile devices involved. The internet connection allows communications via an alternative (or principal) channel.

4.3 Tutor PC Application

As a prototype, the Tutor PC application was designed as a Java application with a simple interface capable of sending and receiving messages to the repository and the Monitored device through the Internet.

4.4 Security Mechanisms

Regarding the security mechanisms, one of the main goals of this work was to assure a secure interaction between the participating parties. Towards this, data confidentiality, authentication and integrity, and freshness must be provided. This is herein provided by end-to-end security, where only the end points need to be considered trustworthy. Additionally, location spoofing is also discussed.

To provide confidentiality, every exchanged message is ciphered by the sender and deciphered by the receiver, using a shared secret key. The SecureTracking system uses the AES algorithm to cipher messages with a key of 128 bits. This algorithm is considered the symmetrical encryption standard, presenting good performance [20,21]. Regarding the cipher mode, CipherText Stealing (CTS) with CBC mode is used, as it does not require padding to match a certain block size. Authentication and integrity validation is provided by the use of Hash-based Message Authentication Code (HMAC) using SHA-256 as hashing function (HMAC-SHA256). The ciphered sent data is appended with the resulting hash value. This hash is obtained from the data itself plus an 8 byte time stamp, also included in the sent data. This time stamp assures message freshness. In this solution it is assumed that the devices have pre-synchronized clocks, configuration that is achieved by synchronizing them with Network Time Protocol (NTP) servers. The used security algorithms were deployed using the `javax.crypto` packages.

Considering the possibility of spoofing attacks on the location system, the implemented solution checks for abrupt changes on the location. Unfortunately, the used platform does not provide many mechanisms to do so. The anti-spoofing mechanisms can be implemented by analyzing the GPS satellites signals and detecting unusual or abrupt changes in their strength.

Considering that the HMAC-SHA256 outputs a MAC value of 32 bytes, and that the timestamp is 8 bytes long, the security overhead in the data transmission is of 40 bytes. A SMS message has 133 bytes that can be used. By allocating 40 bytes for security, a total of 93 bytes are available to be use for the functional logic of the SecureTracking system.

5 Characterization and Evaluation

To evaluate the implementation of the proposed SecureTracking system, the impact of the principal operations was measured using a profiler developed to register the time taken and the battery consumption. The accuracy of the localization methods was also measured. All tests were made in an Android smartphone "LG Optimus Pro C660", running Android OS 2.3.3 with an 800 MHz CPU and 256 MB of RAM.

Considering that every location obtained using the Android API includes an estimation of the accuracy, the localization methods were tested along a predefined path of about 4 km, characterized by both residential and non-residential areas, where the Monitored device was set to obtain its location with an interval of 5 minutes. Figures 3 and 4 present the accuracy estimation, given by the system, for the localization using Network (using Wi-Fi) and A-GPS, where the line represents the actual path. The obtained results suggest that the system was able to locate itself near the real path and that the A-GPS has better accuracy than the Network/Wi-Fi localization.

Since these accuracy values are estimations given by the system, another set of tests was performed to evaluate the accuracy in two different pre-defined

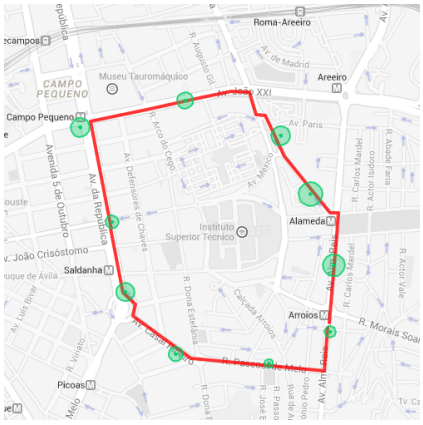


Fig. 3. Accuracy using A-GPS

Fig. 4. Accuracy using Network

paths: one in residential area (Path A) and another in a more industrialized area (Path B). For each path, the location was obtained at four pre-defined locations. The accuracy was manually measured with the assistance of Google Maps to get the coordinates of each fixed point and the Haversine formula (which calculates the shortest distance between two points over the surface of the Earth). All confidence intervals were calculated for a confidence of 90%.

As depicted in Tables 1 and 2, the localization using only Network (Cell-ID) has the worst accuracy values, about 128±48 meters of average error in Path A, and 171±41 meters in Path B. The Wi-Fi based localization registered an average error of 24±12 meters in Path A and 42±12 meters in Path B, being strongly influenced by the density of Wi-Fi APs. Path B had two of the four fixed locations under a dome, which influenced the accuracy error for the A-GPS, as it decreased the line of sight. It registered an average error of 53±18 meters in this path and 21±10 meters in Path A.

Method	Average Distance (m)	Standard Deviation	Variance	Coefficient of Variation	Confidence Interval (m)
A-GPS	21.25	21.43	459.12	1.01	10.17
Wi-Fi	24.83	25.48	649.22	1.03	12.10
Network-only	128.67	102.82	10570.97	0.80	48.82

Table 1. Path A - statistics on the distance between reported and fixed locations

Regarding the time measurements, the localization methods were tested to evaluate the time taken to successfully obtain the location. To perform the A-GPS tests, the assisting data (almanacs and ephemerides) was previously down-

Method	Average Distance (m)	Standard Deviation	Variance	Coefficient of Variation	Confidence Interval (m)
A-GPS	53.06	39.44	1555.22	0.74	18.73
Wi-Fi	42.17	25.75	662.88	0.61	12.23
Network-only	171.25	88.06	7754.02	0.51	41.81

Table 2. Path B - statistics on the distance between reported and fixed locations

loaded to the device before each test. The obtained results are depicted in Table 3. Despite having the assisting data that helps decreasing the TTFF, the A-GPS was the method that took more time, on average, to obtain the location, about 10934 ms ± 4%. The Wi-Fi localization took, on average, almost half of that time, about 5183 ms ± 3%, and obtaining the location using Network only took only about 111 ms ± 5%.

Localization Method	Average (ms)	Standard Deviation	Variance	Coefficient of Variation	Confidence Interval	Max (ms)	Min (ms)
Network-only	111	74.1	5493	0.67	5.45	942	55.6
Wi-Fi	5183	1920	3687631	0.37	141	23489	254
A-GPS	10934	6511	42401337	0.60	479	36711	1783

Table 3. Time statistics for each localization method

The impact of the deployed security mechanisms was also evaluated regarding the time taken to compute the MAC value and to cipher a message. The tests were divided in 10 samples of 1000 tests each, and focused in characterizing the impact of the AES algorithm and the different MAC algorithms (CBC-MAC of 8 bytes and 16 bytes, HMAC-SHA1 and HMAC-SHA256) when processing a typical message of location report. Regarding the MAC computation, the HMAC-SHA256 (used in SecureTracking) was much faster than the CBC-MAC of 16 bytes (3.67±0.07 ms for the first against 8.28 ±0.13 ms on the latter). The obtained results showed that these mechanisms introduce practically no impact considering the update interval expected for these operations. A time delay of $3.67 + 0.07 = 3.74$ ms for the MAC computation using HMAC with SHA256, and $1.23 + 0.021 = 1.25$ ms for the cipher operation using AES-128 in CTS mode, results on an average of $3.74 + 1.25 = 4.99$ ms in a time interval that is expected to be about tens of minutes.

In terms of energy consumption, both the localization methods and the MAC computation were evaluated. During these tests, the battery consumption was measured using the available API in order to read the battery level at the beginning and at end of each sample of tests. To have a proper vision on the impact

of the localization process on the battery lifetime, each localization method was executed the same number of times (500 tests divided in 10 samples of 50 tests each) and with the same interval between consecutive requests (60 seconds). The Wi-Fi localization involves turning the Wi-Fi chipset on (if not active already), keeping it active for some time and scanning for APs. For the Network-only tests, the 3G communication was left on to allow communication with Internet to query Google servers for the location. The Wi-Fi tests were made using known Wi-Fi APs with Internet access to query Google servers. The Wi-Fi chipset was turned off before the test and turned on and off in each location request. The A-GPS tests were made having recent pre-downloaded assisting information about the almanacs and ephemerides. If this was not made, the localization using GPS could take several minutes. All tests included ciphering and computing the MAC value of the message.

Table 4 depicts the obtained results, suggesting that the Network-only localization method consumed more energy than the other two, an average of 4.2±1.36%, against 3.10±0.83% of Wi-Fi and 3.70±0.60% of A-GPS. This is due to the fact that the 3G communication was used to connect to the Internet on the first case. The energy consumption of the Wi-Fi and A-GPS methods are close to each other as the former included turning the Wi-Fi module on in every test and, in the A-GPS tests, the assisting data was pre-downloaded resulting in less time needed to fix at satellites, thus consuming less energy on the localization process. Tests conducted indoors using A-GPS resulted in no fixed satellites at all and no reported location at all in lower floors of buildings. However, locating with A-GPS in the last floor of a building was possible, mostly near windows (with partial line of sight).

Localization Method	Average (%)	Standard Deviation	Variance	Coefficient of Variation	Confidence Interval
Network-only	4.2	2.62	0.07	0.62	1.36
Wi-Fi	3.1	1.60	0.03	0.51	0.83
A-GPS	3.7	1.16	0.01	0.31	0.60

Table 4. Statistics on battery consumption per localization method

Regarding the MAC computation, the battery consumption of CBC-MAC of 16 bytes and HMAC with SHA256 algorithms was also evaluated in order to correlate the time they take to output a MAC value and their energy consumption. The results, depicted in Table 5, suggest that the tests running HMAC-SHA256 (the algorithm used by SecureTracking) consumed less battery (2.11±0.61%, against 4.80±0.97% when running with CBC-MAC16) while providing adequate security strength.

MAC Algorithm	Average decrease (%)	Standard Deviation	Variance	Coefficient of Variation	Confidence Interval
CBC-MAC16	4.8	1.87	0.04	0.39	0.97
HMAC-SHA256	2.11	1.17	0.01	0.55	0.61

Table 5. Battery loss statistics using HMAC-SHA256 and CBC-MAC16

As future work, it can be of interest to improve the robustness of the anti-spoofing measures, for instance, by analyzing the A-GPS/Wi-Fi signal strength. Implementing the option to use dead reckoning could be a valuable feature towards consuming less energy in the location process or as a sanity test for the anti-spoofing mechanism.

6 Conclusion

To develop a secure tracking system, one must not only focus on monitoring the position of a person, but must also provide security mechanisms to assure its correct usage and of the resulting information and to validate the location itself regarding spoofing attacks. This paper proposes the SecureTracking system which provides the means to monitor the position of a person (both periodically and on-demand), applying some self-monitoring to that position using the concept of geofencing. The system also provides end-to-end security on the communication channels maintaining confidentiality, integrity and authentication, and freshness of the exchanged messages. The proposed system also provides an anti-spoofing mechanism that detects abrupt changes on the location reports. The system also includes the definition of SafeSchedules that can be defined by the Tutor, which are periods of time when the Monitored device is considered to be safe and does not need to activate the localization process, thus saving energy. The obtained results suggest a negligible performance impact to the system. In terms of battery consumption, the proposed solution considers a strategy of preferentially choosing the localization method that consumes less energy, namely Wi-Fi, followed by A-GPS if necessary, and by last Cell-ID, if the previous two fail. In conclusion, the system herein proposed shows that a low cost, energy efficient, and secure tracking solution can be deployed using off-the-shelf mobile devices.

References

1. Trevisani, E., Vitaletti, A.: Cell-id location technique, limits and benefits: an experimental study. In: Mobile Computing Systems and Applications, 2004. WMCSA 2004. Sixth IEEE Workshop on, IEEE (2004) 51–60

2. Kos, T., Grgic, M., Kitarovic, J.: Location technologies for mobile networks. In: Systems, Signals and Image Processing, 2007 and 6th EURASIP Conference focused on Speech and Image Processing, Multimedia Communications and Services. 14th International Workshop on, IEEE (2007) 319–322
3. Cavoukian, A., Cameron, K.: Wi-Fi Positioning Systems: Beware of Unintended Consequences Issues Involving the Unforeseen Uses of Pre-existing Architecture. Information and Privacy Commissioner of Ontario, Canada (2011)
4. Mozilla: Mozilla Location Service. https://location.services.mozilla.com/, accessed May 11, 2015.
5. Labs, U.: Unwired Labs Location API. http://unwiredlabs.com, accessed May 11, 2015.
6. Ryoo, J., Kim, H., Das, S.R.: Geo-fencing: geographical-fencing based energy-aware proactive framework for mobile devices. In: Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service, IEEE Press (2012) 26
7. Toorani, M., Beheshti, A.: Solutions to the gsm security weaknesses. In: Next Generation Mobile Applications, Services and Technologies, 2008. NGMAST'08. The Second International Conference on, IEEE (2008) 576–581
8. Hossain, A., Jahan, S., Hussain, M., Amin, M., Newaz, S.S.: A proposal for enhancing the security system of short message service in gsm. In: Anti-counterfeiting, Security and Identification, 2008. ASID 2008. 2nd International Conference on, IEEE (2008) 235–240
9. Ibekwe, I., Aljareh, S.: Sms security: highlighting its vulnerabilities & techniques towards developing a solution. (2012)
10. Tippenhauer, N.O., Rasmussen, K.B., Pöpper, C., Čapkun, S.: Attacks on public wlan-based positioning systems. In: Proceedings of the 7th international conference on Mobile systems, applications, and services, ACM (2009) 29–40
11. Warner, J.S., Johnston, R.G.: Gps spoofing countermeasures. Homeland Security Journal (2003)
12. Life360: GPS Tracker. https://www.life360.com/gps-tracking/, accessed October 1, 2014.
13. greenalp: Real Time GPS Tracker. http://www.greenalp.com/RealTimeTracker/, accessed October 1, 2014.
14. AD, M.I.: MobileMonitor. http://www.mobilemonitor.com, accessed September 30, 2014.
15. Bulut, M., Fatih, D., Murat: Energy efficient proximity alert on android. In: Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on, IEEE (2013) 157–162
16. Wang, H., Wang, Z., Shen, G., Li, F., Han, S., Zhao, F.: Wheelloc: Enabling continuous location service on mobile phone for outdoor scenarios. In: INFOCOM, 2013 Proceedings IEEE, IEEE (2013) 2733–2741
17. Papandrea, M., Giordano, S.: Enhanced localization solution. In: Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on, IEEE (2012) 241–246
18. Alzantot, M., Youssef, M.: Uptime: Ubiquitous pedestrian tracking using mobile phones. In: Wireless Communications and Networking Conference (WCNC), 2012 IEEE, IEEE (2012) 3204–3209
19. Bilgic, H.T., Alkar, A.: A secure tracking system for gps-enabled mobile phones. In: Information Technology and Multimedia (ICIM), 2011 International Conference on, IEEE (2011) 1–5

20. Rizvi, S., Hussain, S.Z., Wadhwa, N.: Performance analysis of aes and twofish encryption schemes. In: Communication Systems and Network Technologies (CSNT), 2011 International Conference on, IEEE (2011) 76–79
21. Soni, S., Agrawal, H., Sharma, M.: Analysis and comparison between aes and des cryptographic algorithm. International Journal of Engineering and Innovative Technology **2** (2012) 175–191

PATH - Visualização de Percursos Pessoais com Mapas Animados em Dispositivos Móveis

Tiago Gonçalves, Ana Rita Vieira, Ana Paula Afonso, António Ferreira

LaSIGE, Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa,
1749-016 Lisboa, Portugal
tgoncalves@lasige.di.fc.ul.pt,
vieira.arc@gmail.com,{apafonso,amferreira}@fc.ul.pt

Resumo. Com a crescente popularidade de dispositivos móveis (como *smart-phones* e *tablets*) e de aplicações que capturam e armazenam dados geográficos, cada vez mais pessoas gravam as suas deslocações sob a forma de dados de trajetórias. Este padrão emergente é exemplificado com o aumento do uso de aplicações que, para além de gravarem a evolução da trajetória seguida pelo utilizador, permitem a visualização desses mesmos dados, tipicamente sobre a forma de mapas estáticos 2D, complementados com vários diagramas de modo a extrair conhecimento dos dados. Apesar dos vários estudos na área da visualização de dados espácio-temporais, a sua aplicação em dispositivos móveis para representação de dados de trajetos pessoais ainda se encontra pouco explorada. Os mapas animados têm emergido como uma potencial técnica para a visualização de informação de forma dinâmica e são intuitivos na deteção de relações entre a informação espacial e temporal. Este artigo tem como objetivo apresentar o PATH, um protótipo de uma aplicação de visualização de trajetos pessoais utilizando mapas animados, com o intuito de explorar e avaliar a adequabilidade desta técnica na representação de trajetos de atividades físicas em dispositivos móveis.

Palavras chave: Visualização de informação, mapas animados, dispositivos móveis, trajetórias pessoais, interação pessoa-máquina.

1 Introdução

Com o crescente desenvolvimento e utilização de serviços baseados em localização, particularmente em dispositivos móveis, e a crescente disponibilidade comercial de vários tipos de sensores, é cada vez mais fácil e comum a recolha de grandes volumes de dados espácio-temporais, representativos de trajetos, ações, e desempenho dos utilizadores [1]. Como consequência somos, cada vez mais, expostos a grandes quantidades de informação, não necessariamente relacionada com os nossos interesses profissionais mas, ainda assim, relevantes em contextos pessoais [2]. Um exemplo deste padrão emergente consiste no aumento da popularidade de aplicações para dispositivos móveis, tais como o Endomondo[1] ou o Runkeeper[2] que, para além de

[1] https://www.endomondo.com/

gravarem a evolução das trajetórias seguidas pelos utilizadores, permitem visualizar esses mesmos dados complementados com várias estatísticas, tais como diferenças de velocidade praticadas ou o número estimado de calorias gasto, entre outras. Consequentemente, existe um enorme potencial para cada um de nós utilizar estes dados para variar e/ou melhorar as nossas atividades diárias. Para tal, são necessárias visualizações adequadas, que permitam uma análise e compreensão adequada dos dados.

Uma trajetória consiste na evolução das propriedades espaciais de um objeto ao longo do tempo [3]. Tipicamente, estas são representadas como uma sequência de pontos ordenados temporalmente, em que cada ponto, ou conjunto de pontos, contém informação sobre a sua localização espacial (onde foi detetado), a sua localização temporal (quando foi detetado) e um conjunto de atributos, denominados temáticos, que permitem caraterizar esse ponto (por exemplo, a velocidade de deslocação). Devido às caraterísticas espaciais associadas às trajetórias, os mapas são considerados métodos fundamentais para a sua visualização e análise [4]. Em particular, os mapas animados têm surgido como uma potencial técnica para a visualização de informação espácio-temporal, dada a sua eficácia para a perceção de variações de informação ao longo do tempo, assim como para a comparação entre dados relacionados [5]. Consequentemente, o estudo de técnicas de visualização e interação para a análise de dados espácio-temporais em contextos pessoais e a análise da adequabilidade dos mapas animados são dois desafios relevantes, em particular devido aos problemas de usabilidade inerentes ao tamanho reduzido do ecrã dos dispositivos móveis. Contudo, apesar do crescente número de aplicações de visualização de dados espácio-temporais, o estudo da sua usabilidade tende a ser pouco explorado [2, 13, 14], por exemplo com utilizadores não peritos, não familiarizados com a análise de dados espácio-temporais, e que, no entanto, são os utilizadores deste tipo de aplicações [2, 15].

Este artigo tem como objetivo abordar estes desafios. Para isso, é apresentado o PATH, *Personal Animated Trajectory Helper*, um protótipo de uma aplicação de visualização de trajetórias pessoais recorrendo a mapas animados, com o intuito de explorar e avaliar a adequabilidade desta técnica na representação de trajetos de atividades físicas em dispositivos móveis.

Este artigo está organizado da seguinte forma: na Secção 2 são identificados os principais trabalhos relacionados; na Secção 3 é apresentado o protótipo PATH e as suas principais funcionalidades; na Secção 4 é descrito o processo de avaliação do PATH, e na Secção 5 são apresentadas as principais conclusões e o trabalho futuro.

2 Trabalho Relacionado

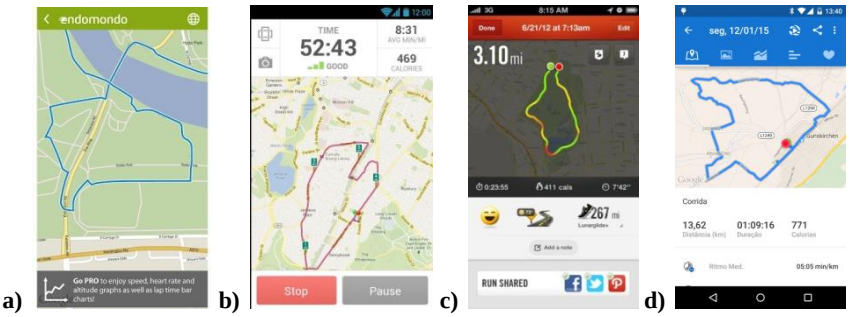
Os mapas geográficos são considerados essenciais para a visualização de trajetórias devido às propriedades espaciais dos seus dados [4]. Em particular, os mapas estáticos em duas dimensões (2D) estão entre as técnicas mais usadas para a representação de informação georreferenciada. Tipicamente, pontos e/ou linhas são usados sobre um

[2] http://runkeeper.com/

plano/mapa para representar a evolução da posição de um objeto ao longo do tempo, enquanto outros símbolos e/ou variáveis visuais associadas a símbolos existentes, como, a cor, o tamanho, ou a forma de um ponto, podem ser usados para representar outros tipos de informação temática (por exemplo, velocidade, direção, ou elevação; ver Figura 1) [9].

Apesar do crescente desenvolvimento de tecnologias para a visualização de informação em três dimensões (mapas em 3D), estudos anteriores sugerem que a utilização deste tipo de mapas tem limitações do ponto de vista percetual e pode causar desconforto ao utilizador [16, 17]. Por outro lado, os mapas 2D continuam a ser mais comuns e costumam ser preferidos face a mapas 3D [12], sendo plausível assumir que utilizadores não peritos estarão mais familiarizados com um mapa em 2D. Finalmente, devido ao tamanho reduzido do ecrã do dispositivo, a utilização de mapas 3D em dispositivos móveis ainda é um tópico pouco explorado, o que pode levantar várias questões e/ou restrições e, como tal, não fazem parte dos objetivos deste estudo.

Por sua vez, mapas 2D são cada vez mais utilizados em aplicações para recolha e visualização de informação pessoal em dispositivos móveis. Aplicações como Endomondo, RunKeeper, NikePlus[3], Runtastic[4] (ver Figura 1), entre outras, permitem a qualquer tipo de utilizador visualizar as suas trajetórias, e recolher algumas estatísticas, tais como a velocidade ou a altitude/elevação do percurso, entre outras. Contudo, este tipo de mapas quando utilizado em dispositivos móveis tende a não explorar a componente temporal dos dados, tipicamente necessitando de simbologia e/ou visualizações adicionais [13]. Por esse motivo, os mapas animados são sugeridos como uma alternativa a mapas estáticos.



a) **b)** **c)** **d)**

Fig. 1. Visualização de trajetórias em mapas 2D para dispositivos móveis em diversas aplicações: **a)** Endomondo; **b)** RunKeeper; **c)** Nikeplus – neste exemplo, em particular, a cor é utilizada para representar a velocidade (verde – lento, vermelho – rápido); e **d)** Runtastic.

Um mapa animado consiste na representação de uma sequência de mapas (*frames*), normalmente numa vista única, tomando partido das capacidades computacionais dos dispositivos em atualizarem os seus conteúdos rapidamente (ver Figura 2). Podem ser consideradas duas categorias de mapas animados: *temporais*, se os eventos

[3] https://nikeplus.com/
[4] https://www.runtastic.com

apresentados na animação ocorrem por ordem cronológica e/ou se existe uma relação entre o tempo de animação e o tempo real; *não-temporais*, caso as animações sejam utilizadas para enfatizar diferenças na representação, sem qualquer relação com o tempo real (por exemplo, o uso de intermitência num símbolo para capturar a atenção do utilizador) [14]. No contexto deste trabalho, o foco é colocado nas animações temporais. É importante também referir que, no contexto deste estudo, estamos a usar o termo mapa *estático* em oposição a mapa *animado* e não em oposição a mapa *interativo* [15].

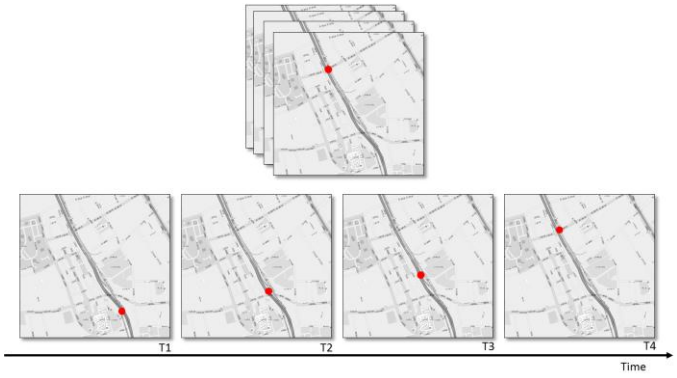


Fig. 2. Representação de um mapa animado. A visualização é composta por vários mapas, cada um apresentado num instante de tempo específico.

Os mapas animados, quando comparados com os mapas estáticos, possuem uma dimensão adicional, o tempo da animação, sobre a qual é possível apresentar informação, permitindo que sejam usadas outras variáveis visuais, como a cor e o tamanho, para representar e/ou realçar outros atributos. Contudo, a quantidade de informação a memorizar tenderá a aumentar proporcionalmente à duração da animação, o que poderá criar problemas de perceção, dado que quanto maior a duração da animação, menor será a capacidade de memorizar informação relevante. Consequentemente, existem estudos que sugerem que o tempo adequado de exposição de um utilizador a uma animação não deve exceder um minuto [14]. Por outro lado, existem outros estudos que sugerem que o uso de animação é adequado para a deteção e identificação de caraterísticas e padrões de movimento, como paragens e mudanças de velocidade [4, 7, 12]. Adicionalmente, a utilização de animação foi considerada mais apelativa para o utilizador quando comparada com as representações estáticas [18]. Isto poderá ser importante na representação de informação em contextos pessoais, uma vez que as ferramentas de visualização deste tipo de dados podem ser usadas por uma variedade de utilizadores em contextos de utilização que deverão requerer uma carga cognitiva tão pequena quanto possível, como quando querem *ocupar o tempo*, ou estão simplesmente *curiosos*, ou mesmo *entediados* [2].

Para responder a estes desafios, na secção seguinte apresenta-se o PATH, um protótipo de uma aplicação de visualização de trajetórias pessoais, recorrendo a mapas animados em dispositivos móveis.

3 PATH para a Visualização de Percursos Pessoais

De modo a explorar e avaliar a adequabilidade de mapas animados em dispositivos móveis para a visualização de dados espácio-temporais em contextos pessoais, desenvolvemos o protótipo PATH, representado na Figura 3.

Dada a crescente popularidade e utilização de aplicações como o Endomondo, ou o NikePlus, optámos por usar dados de trajetos pessoais na prática de atividades físicas, por exemplo, caminhar ou correr, como principal caso de estudo.

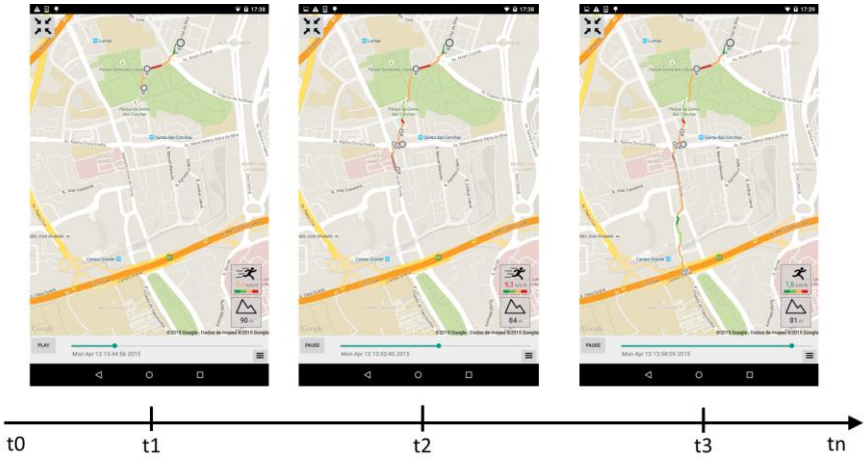


Fig. 3. PATH - Visualização de um trajeto pessoal em três instantes de tempo (t1, t2 e t3).

A animação apresentada pelo protótipo PATH (Figura 3) consiste no desenho progressivo do trajeto seguido por um dado utilizador ao longo do tempo num mapa 2D.

Através do desenho do trajeto seguido pelo utilizador e complementando essa informação com um conjunto de símbolos e/ou cores representativas de diferentes atributos associados ao movimento, é disponibilizada a recolha de informação sobre duas perspetivas temporais: o *instante atual* de movimento, isto é, informação sobre o ponto em movimento durante o tempo corrente da animação; e o *período total* de movimento, ou seja, informação passada, mas que pode ser comparada com a do instante atual ou de um outro instante no passado.

Sobre estas duas perspetivas temporais, o PATH disponibiliza quatro tipos de informação: a trajetória seguida, a elevação/altitude, a direção de movimento, e a velocidade. Nas secções seguintes são descritas as respetivas representações.

3.1 Trajetória Seguida

Para a representação da trajetória seguida na animação, um símbolo do utilizador é constantemente alterado em função da sua posição relativa ao *instante atual*. Em cada instante de tempo, o símbolo deixa um *rasto*, sobre a forma de uma linha e/ou uma

cópia do símbolo indicando a evolução do trajeto ao longo do *período total* do tempo de animação (ver Figura 3).

3.2 Elevação/Altitude

Na visualização de mapas 2D, o atributo elevação (ou altitude) está, naturalmente, relacionado com as coordenadas espaciais de pontos da trajetória e, logo, com a evolução da posição do utilizador. Contudo, quando a visualização tem duas dimensões, não é possível utilizar a variável visual *posição* para representar a elevação, pelo que decidiu-se considerar este tipo de informação como um atributo temático quantitativo. Deste modo, recorreu-se a três estratégias para apresentar a elevação/altitude:

- Símbolo de montanha sobre o mapa: este símbolo apresenta a altitude durante o *instante atual* da animação e está localizado no canto inferior direito da visualização (Figura 4a);
- Variável visual *tamanho*: quanto maior o símbolo, neste caso um circulo, maior a altitude em que o utilizador se encontrou (Figura 4b);
- Símbolo de *hambúrguer*: quanto maior o número de ícones/linhas por símbolo, maior a altitude nessa localização (Figura 4c).

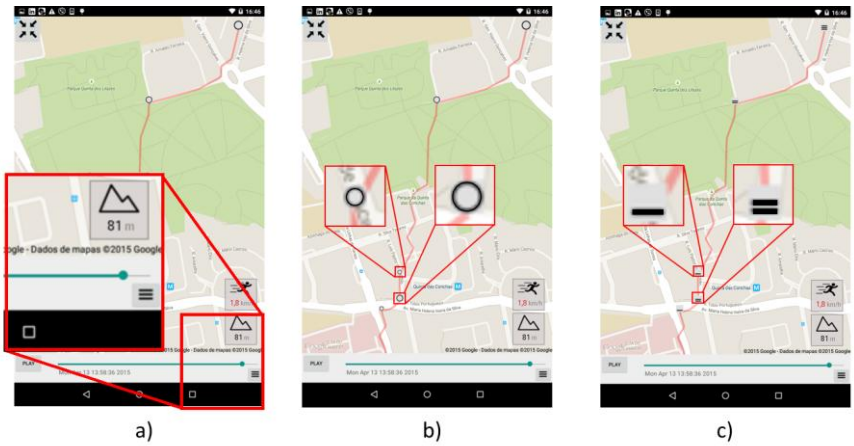


Fig. 4. Representação da altitude/elevação de um trajeto no PATH através de: **a)** símbolo de montanha para representação da altitude atual; **b)** tamanho do símbolo (círculo) em pontos no trajeto; e **c)** número de linhas no ícone de *hambúrguer*.

O primeiro método tem a vantagem de fornecer informação precisa em cada instante de tempo. Contudo, recorre à capacidade do utilizador em memorizar informação ou à utilização de métodos de pesquisa interativos para permitir a comparação dessa informação em vários instantes de tempo. Além disso, é um ícone separado daquele que representa o movimento atual do utilizador, pelo que poderá passar despercebido. Por outro lado, o segundo e o terceiro métodos embora não

apresentem valores de altitude de forma detalhada, permitem adquirir uma noção das diferenças de altitude em diferentes locais e instantes do tempo.

3.3 Direção e Velocidade do Movimento

Devido às caraterísticas intrínsecas dos mapas animados e de forma semelhante à evolução do trajeto seguido pelo utilizador, a direção e a velocidade são implicitamente fornecidas através da direção e velocidade a que o símbolo correspondente ao estado atual do utilizador se move no mapa. Contudo, para realçar essa informação e permitir a sua comparação com instantes de tempo passados, foi decidido recorrer a simbologia adicional.

Para a representação da direção, são usadas setas em cada ponto da trajetória, apontando na direção seguida pelo utilizador (Figura 5a). Para a representação da velocidade foram consideradas duas opções:

- Escala de cores: utilizando como referência aplicações já existentes, como o Nikeplus, escolhemos um conjunto de cores indicativas do esforço praticado durante o movimento. Deste modo, as cores mais próximas de vermelho representam velocidades elevadas e as cores mais próximas de verde representam velocidades mais reduzidas (Figura 5a);
- Extensão do símbolo representativo da direção: quanto maior o número de extensões/setas, mais rápido o movimento representado (Figura 5b).

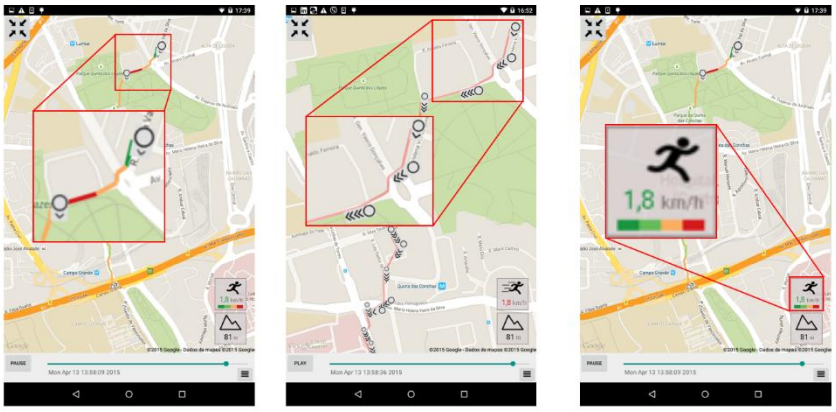


Fig. 5. Representação da velocidade de um trajeto no PATH através de: **a)** escala de cores; **b)** número de símbolos; e **c)** símbolo para representação da velocidade atual.

Recorrendo a estas duas opções de representação, é possível detetar, como visível nas Figuras 5a e 5b, uma aceleração do utilizador nos instantes iniciais do trajeto. Adicionalmente, à semelhança da representação da elevação durante o instante atual da animação, foi decidido utilizar um símbolo adicional, no canto inferior direito do ecrã, para representar a informação sobre a velocidade atual do utilizador (Figura 5c).

Embora a utilização de setas adicionais para realçar a representação da velocidade possa ser mais facilmente compreensível devido à relação crescente entre o número de ícones e a velocidade representada, esta é uma alternativa que promove um maior *ruído visual* na representação. De facto existe a possibilidade de vários pontos do trajeto terem velocidades significativamente diferentes (por exemplo, quando um utilizador alterna entre andar devagar e correr) o que pode aumentar o número de símbolos no ecrã. Por outro lado, a utilização de cores para a representação da velocidade não requer simbologia adicional, minimizando a possibilidade de um grande *ruído visual* na representação. Contudo, pode ser considerada menos expressiva relativamente aos valores a que correspondem, uma vez que é necessário que o utilizador associe significado às diferentes cores.

3.4 Controlos e Configurações Adicionais

O protótipo PATH é composto por três componentes principais com as quais o utilizador pode interagir (Figura 6): o mapa, a *timeline* e o modo de controlo da animação.

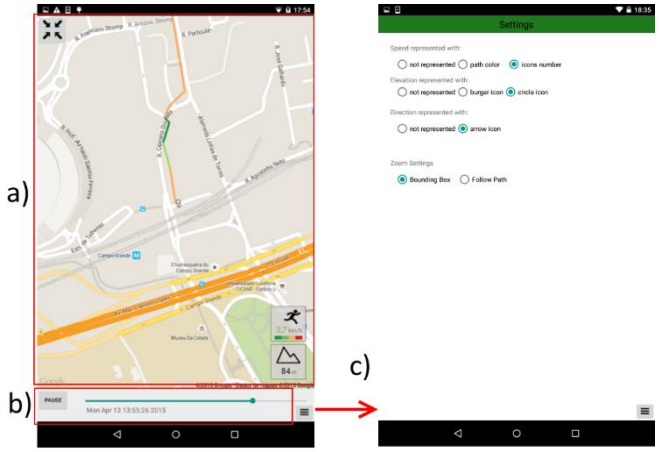


Fig. 6. Componentes principais do PATH: **a)** mapa 2D; **b)** *timeline*; e **c)** menu de configurações.

O mapa é a componente principal onde é apresentada a informação ao utilizador (Figura 6a). Esta permite a realização de operações de *panning* e *zooming*, de modo a, respetivamente, arrastar e alterar o nível de detalhe do mapa. Por outro lado, permite também a recolha de informação sobre a velocidade e a elevação detetada a cada ponto da trajetória, fazendo clique no símbolo respetivo.

A *timeline* encontra-se desenhada na parte inferior do ecrã (Figura 6b) e permite ao utilizador começar e parar a animação (com o botão *Play/Pause*), mostrar o instante de tempo real correspondente ao momento atual da animação, através da informação textual. Adicionalmente, o *slider* permite obter informação aproximada sobre a duração já passada da animação e até ao final da mesma. Por sua vez, esta

funcionalidade, permite também ao utilizador avançar ou recuar no tempo da animação.

Finalmente, o menu de configurações (Figura 6c) suporta as seguintes operações:

- Seleção e combinação das representações de atributos que deverão ser utilizadas para apresentar informação (descritas nas secções anteriores);
- Definição do foco da animação segundo duas categorias:
  - *foco no instante atual* (Figura 7a): o mapa está sempre centrado no símbolo que representa o *instante atual* da animação. Esta estratégia realça a visualização do estado atual de um utilizador durante uma animação;
  - *foco no trajeto total* (Figura 7b): o nível de *zoom* e o centro do mapa são ajustados, antes do início da animação, de modo a que a área geográfica em que foi detetado movimento esteja completamente visível no ecrã. Esta estratégia realça a visualização de uma animação com o foco em todo o movimento detetado.

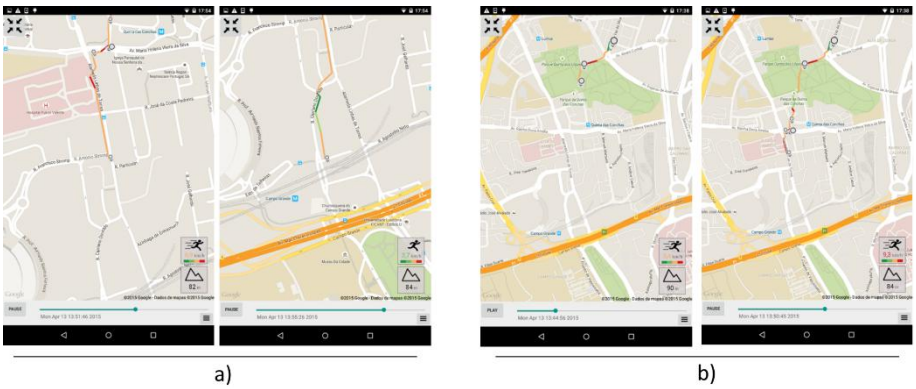


Fig. 7. Visualização de um trajeto no PATH com foco: **a)** no instante atual; e **b)** no trajeto total.

4 Metodologia de Avaliação

Para a avaliação das funcionalidades descritas no protótipo PATH está em curso um estudo com utilizadores, dividido em duas etapas. A primeira etapa tem como objetivo recolher a opinião dos utilizadores sobre os símbolos utilizados para as representações estáticas da *velocidade* e *elevação* e a segunda tem como finalidade analisar a adequabilidade do uso da técnica de animação para a representação de dados espácio-temporais em contextos de visualização pessoal.

4.1 Recolha das Opiniões de Utilizadores

A primeira etapa da avaliação engloba a realização de um estudo piloto no qual os participantes foram expostos às diversas alternativas de representação dos atributos descritos nas secções anteriores. Esta etapa serviu para identificar quais as

representações preferidas dos participantes, de forma a reduzir o número de variáveis em análise nos estudos de usabilidade. Isto é importante pois menos variáveis conduzem a uma diminuição da quantidade de tarefas (e animações) a que cada participante é exposto e, deste modo, minimizam-se possíveis efeitos de cansaço e aprendizagem.

O estudo piloto contou com a participação de 10 voluntários, com idades entre os 16 e os 56 anos (média = 29.9; desvio padrão = 13.9). Todos os participantes, embora não peritos em análise espácio-temporal, tinham alguma experiência com interação e visualização de informação em dispositivos móveis. Em particular, 3 desses participantes estão também familiarizados com aplicações de registo de atividades pessoais em dispositivos móveis.

No início do estudo foram explicados, a cada participante, os objetivos do trabalho e apresentadas as alternativas de representação para os atributos de *velocidade* e *elevação.* De seguida, foi disponibilizado um período de tempo livre a cada participante, para que este analisasse e comparasse as diferentes alternativas. A Figura 8 apresenta o resumo dos resultados obtidos neste estudo piloto.

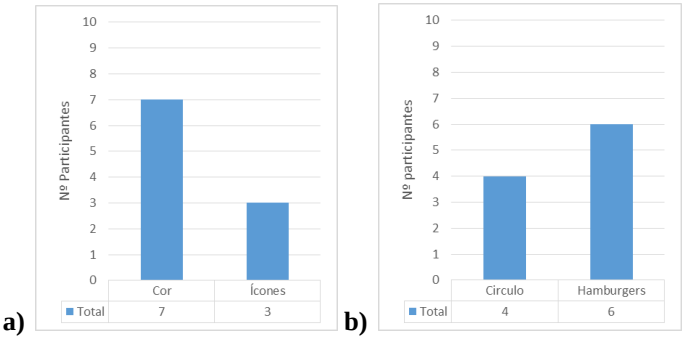


Fig. 8. Preferências dos participantes relativamente aos diferentes tipos de representação dos atributos de: **a)** *velocidade*; e **b)** *elevação.*

Globalmente, os participantes mostraram uma maior preferência por representações que ocupam menos espaço na visualização e que são menos suscetíveis de se sobreporem, nomeadamente, a representação da *velocidade* do percurso através do atributo *cor* e a representação da *elevação* do percurso através do ícone de *hambúrguer*. Em particular, para a visualização da elevação, alguns participantes comentaram que acharam este método mais intuitivo, por ser mais fácil reconhecer grandes diferenças de altitude, isto é, determinar quais os locais mais ou menos elevados do trajeto. Com base nestes resultados, na próxima etapa de avaliação serão apenas consideradas as combinações das representações preferidas pelos participantes.

4.2 Estudo de Usabilidade

A segunda etapa da avaliação do PATH consiste num estudo de usabilidade cujo objetivo é analisar a adequabilidade dos mapas animados para representar dados

espácio-temporais em contextos pessoais. Para isso, foram consideradas três variantes:

- Mapa estático 2D: esta variante apresenta um mapa em 2D estático que permite pesquisar por informação em qualquer instante de tempo, através da utilização de um *slider* temporal;
- Mapa animado com *foco no momento atual*: ver Figura 7a;
- Mapa animado com *foco no trajeto total*: ver Figura 7b.

Deste modo, pretende-se comparar a adequabilidade do uso de diferentes tipos de animação comparativamente com uma variante estática (ainda que interativa). Tendo em conta as propriedades de cada variante, as nossas hipóteses são as seguintes:

1. Os mapas animados, em particular com o *foco no trajeto total*, serão preferidos aos mapas estáticos;
2. Os mapas animados serão, de uma forma geral, mais adequados para ajudar o participante a interpretar a informação apresentada;
3. Os mapas estáticos serão mais adequados para ajudar o participante a recolher/analisar informação detalhada.

De modo a testarmos estas hipóteses, os participantes serão expostos a duas tarefas, baseadas nos tipos mais comuns de tarefas de visualização cartográfica presentes na literatura [19]:

- Identificação/descrição: considerando um conjunto de *keywords*, o participante deverá descrever em termos de velocidade e de direção um trajeto visualizado nos protótipos. As *keywords* utilizadas consistem em pequenas afirmações que o participante deverá completar, de modo a descrever o movimento observado. Por exemplo, "O movimento foi de *x* para *y"*, tal que x e y podem ser: *cima, baixo, esquerda e direita;* A velocidade de movimento foi *constante, crescente, decrescente*, ou *variável*;
- Comparação: considerando um determinado percurso, o participante deverá determinar em que locais o trajeto foi efetuado com maior velocidade e/ou determinar os locais de maior altitude.

Em ambas as tarefas os participantes dispõem de tempo para analisarem livremente um trajeto pessoal. Após este período, na primeira tarefa, o participante deverá completar as frases descritivas do percurso com a lista de *keywords*. Na segunda tarefa, e após a visualização da animação, será apresentado um mapa estático sem qualquer informação temática (ou seja, sem informação relativa à velocidade, direção e altitude dos pontos do percurso) e no qual o participante terá de responder às questões apresentadas nas tarefas.

Neste estudo é considerada uma variável independente, técnica de visualização, composta por três níveis, correspondentes às três variantes do protótipo PATH descritas anteriormente, ou seja: mapa estático, mapa animado com foco no instante atual, e mapa animado com foco no trajeto total.

A experiência será realizada segundo uma configuração de *within-subjects*, em que cada participante realizará as tarefas com as variantes individualmente. No início da

experiência, serão explicados os principais objetivos do estudo a cada participante, será apresentada uma demonstração de cada variante, e será possível ao mesmo interagir com cada variante, de modo a familiarizar-se com os controlos disponibilizados e esclarecer qualquer dúvida. Após a fase de treino, os participantes realizarão as duas tarefas com as três variantes de representação. A ordem pela qual as variantes serão apresentadas aos participantes será balanceada usando um *quadrado latino* (*Latin square*). Consequentemente, cada participante terá de realizar um total de 6 tarefas: 3 visualizações x 2 tarefas.

Para avaliar as hipóteses formuladas serão consideradas as seguintes variáveis dependentes:

- Tempo: tempo despendido pelo participante para realizar as tarefas, após o período inicial de análise;
- Erros/Precisão: na tarefa de identificação/descrição, um erro corresponde a uma *keyword* escolhida incorretamente ou a uma *keyword* correta não escolhida. Na tarefa de comparação, a precisão será recolhida em função da distância do local assinalado pelo participante à localização correta. Quanto maior for a distância obtida, maior será o erro considerado;
- Preferências: no final de cada tarefa, cada participante terá de classificar as três variantes relativamente à sua utilidade para a realização das tarefas numa escala de *Likert*;
- Número de ações: esta variável será recolhida através do número de operações de arrastamento e mudança de escala (*panning/zooming*), e do número de ações sobre os controlos temporais das variantes (por exemplo, arrastamento da *timeline* e (re)começar a animação). Esta métrica pretende complementar a métrica *tempo* de modo a analisar que tipo de técnicas são mais complexas para o participante e/ou requerem maior interação, e perceber de que modo este utiliza as caraterísticas de pesquisa temporal nos mapas.

No final de cada tarefa, será também pedido ao utilizador que forneça as suas opiniões relativamente às variantes, de modo a perceber quais as dificuldades encontradas, os aspetos mais positivos e aqueles que devem ser melhorados de modo a auxiliá-lo a completar as tarefas propostas.

5 Conclusões e Trabalho Futuro

Este artigo apresenta o PATH, um protótipo de uma aplicação de visualização de trajetórias recorrendo a mapas animados, com o intuito de explorar os desafios de desenho e a adequabilidade desta técnica na representação de trajetos pessoais em dispositivos móveis.

Estão em curso estudos com utilizadores para a avaliação das funcionalidades propostas no PATH. Em particular, foi realizado um estudo piloto com 10 participantes de modo a recolher opiniões relativamente aos tipos de representações preferidas para os diferentes atributos temáticos associados ao movimento. Os resultados sugerem que, embora com uma pequena margem de diferença, os participantes tendem a preferir representações que ocupem menos espaço, que evitem

possíveis sobreposições de informação e permitam uma melhor comparação entre valores muito distintos de um determinado atributo (por exemplo, locais mais ou menos elevados).

Como trabalho futuro iremos otimizar o protótipo e proceder à sua avaliação de acordo com os resultados obtidos no estudo piloto. Em particular, iremos realizar um estudo de usabilidade de modo a comparar dois tipos diferentes de mapas animados com uma representação estática, de modo a determinar as possíveis des/vantagens deste tipo de mapas tendo em conta um público alargado de utilizadores.

References

1. Amini, F., Rufiange, S., Hossain, Z., Ventura, Q., Irani, P., McGuffin, M.J.: The Impact of Interactivity on Comprehending 2D and 3D Visualizations of Movement Data. IEEE Trans. Vis. Comput. Graph. 21, 122–135 (2015).
2. Huang, D., Tory, M., Aseniero, B.A., Bartram, L., Bateman, S., Carpendale, S., Tang, A., Woodbury, R.: Personal Visualization and Personal Visual Analytics. Vis. Comput. Graph. IEEE Trans. 21, 420–433 (2015).
3. Lee, W.-C., Krumm, J.: Trajectory Preprocessing. In: Zheng, Y. and Zhou, X. (eds.) Computing with Spatial Trajectories. pp. 3–33. Springer (2011).
4. Kraak, M.-J.: Geovisualization and Time – New Opportunities for the Space–Time Cube. In: Dodge, M., McDerby, M., and Turner, M. (eds.) Geographic Visualization Concepts, Tools and Applications. pp. 294–306. Wiley (2008).
5. Demissie, B.: Geo-Visualization of Movements: Moving Objects in Static Maps, Animation and The Space-Time Cube. VDM Publishing (2010).
6. Chen, C.: Top 10 Unsolved Information Visualization Problems. IEEE Comput. Graph. Appl. 25, 12–16 (2005).
7. Elmqvist, N., Yi, J.S.: Patterns for Visualization Evaluation. In: Proceedings of the 2012 BELIV Workshop: Beyond Time and Errors - Novel Evaluation Methods for Visualization. pp. 12:1–12:8 (2012).
8. Andrienko, G., Andrienko, N., Demsar, U., Dransch, D., Dykes, J., Fabrikant, S.I., Jern, M., Kraak, M.-J., Schumann, H., Tominski, C.: Space, time and Visual Analytics. Int. J. Geogr. Inf. Sci. 24, 1577–1600 (2010).
9. Bertin, J.: Semiology of Graphics: Diagrams, Networks, Maps. University of Wisconsin Press (1967).
10. Seipel, S., Carvalho, L.: Solving Combined Geospatial Tasks Using 2D and 3D Bar Charts. In: Proceedings of 16th International Conference on Information Visualisation. pp. 157–163 (2012).
11. John, M.S., Cowen, M.B., Smallman, H.S., Oonk, H.M.: The use of 2D and 3D displays for shape-understanding versus relative-position tasks. Hum. Factors J. Hum. Factors Ergon. Soc. 43, 79–98 (2001).
12. Goncalves, T., Afonso, A.P., Martins, B.: Visualizing Human Trajectories: Comparing Space-Time Cubes and Static Maps. In: Proceedings of HCI 2014. pp. 207–212 (2014).
13. 13. Andrienko, N., Andrienko, G., Bak, P., Keim, D., Stefan, W.: Visual Analytics of Movement. Springer (2013).
14. Harrower, M., Fabrikant, S.: The Role of Map Animation for Geographic Visualization. In: Dodge, M., McDerby, M., and Turner, M. (eds.) Geographic Visualization Concepts, Tools and Applications. pp. 49–65. Wiley (2008).
15. Roth, R.E.: Cartographic Interaction: What we know and what we need to know. J. Spat. Inf. Sci. 59–115 (2013).

16. Slocum, T., Sluter, R., Kessler, F., Yoder, S.: A Qualitative Evaluation of MapTime, A Program For Exploring Spatiotemporal Point Data. Cartogr. Int. J. Geogr. Inf. Geovisualization. 39, 43–68 (2004).
17. Willems, N., van de Wetering, H., van Wijk, J.J.: Evaluation of the Visibility of Vessel Movement Features in Trajectory Visualizations. Comput. Graph. Forum. 801–810 (2011).
18. Robertson, G., Fernandez, R., Fisher, D., Lee, B., Stasko, J.: Effectiveness of Animation in Trend Visualization. IEEE Trans. Vis. Comput. Graph. 14, 1325–1332 (2008).
19. Roth, R.E.: Cartographic Interaction Primitives: Framework and Synthesis. Cartogr. J. 376–395 (2012).

Enriquecimento de plataformas web colaborativas com comunicação browser-a-browser

Albert Linde, João Leitão e Nuno Preguiça

NOVA LINCS & DI-FCT-Universidade Nova de Lisboa

Resumo A popularidade das aplicações colaborativas na Web tem crescido significativamente nos últimos anos. Estas incluem ferramentas de edição como o Google Docs, Microsoft Office 365, assim como outros tipos de plataformas Web como as redes sociais. Apesar de todas estas aplicações serem centradas nos utilizadores, estas continuam a recorrer a um modelo de interação mediado por uma componente centralizada, que, para além de ser um ponto de contenção do qual depende todas as interações entre os clientes, também leva a que a propagação de conteúdos entre os utilizadores seja afetada por grandes latências de comunicação. Para ultrapassar estas limitações, propomos enriquecer as arquiteturas atuais com suporte para comunicação direta browser-a-browser. Para tal, apresentamos o desenho de um sistema que permite a aplicações a executarem em múltiplos browsers manterem réplicas de um conjunto de objetos, os quais podem modificar concorrentemente. Os browsers propagam as modificações efetuadas diretamente usando WebRTC, recorrendo a um servidor como intermediário quando tal não é possível. A convergência das réplicas é alcançada usando uma solução baseada em CRDTs. Para demonstrar os benefícios da solução proposta, implementámos um sistema que expõe uma API semelhante ao Google Drive Realtime, e apresentamos resultados experimentais que suportam a nossa proposta.

1 Introdução

Nos últimos anos temos assistido à proliferação e ao aumento da popularidade de aplicações web. Muitas destas aplicações são centradas em utilizadores, e muitas apresentam aspetos colaborativos ou proporcionam suporte para a interação directa entre os seus utilizadores. Exemplos relevantes destas aplicações são as redes sociais, aplicações de troca de mensagens escritas, ou ferramentas de produtividade colaborativas como o GoogleDocs ou o Mircrosoft Office 365.

A complexidade destas aplicações tem vindo a aumentar, usufruindo das novas funcionalidades e capacidades dos *browsers*. De facto, e graças a popularidade crescente de linguagens como Javascript, muitas destas aplicações executam a maior parte do seu código diretamente no browser do utilizador.

No entanto, e apesar do aumento significativo das capacidades dos browsers e da natureza centrada no utilizador de uma parte significativa destas aplicações, estas continuam assentes sobre o modelo cliente-servidor, no qual toda a interação entre os utilizadores é mediada através de um servidor centralizado. As limitações do modelo cliente-servidor são conhecidas e incluem: i) o facto de incorrer em latências significativas entre utilizadores, especialmente aqueles que se encontram geograficamente próximos (ou até numa única rede local de uma empresa ou outra instituição); ii) o

servidor central é um potencial ponto de contenção que pode limitar a escalabilidade do sistema em termos de número de utilizadores que podem ser servidos devido ao limite de recursos na componente centralizada; e iii) o servidor constituí um ponto único de falha que impede qualquer interação entre os utilizadores no caso em que o servidor deixa de estar disponível.

Factualmente, o advento da computação em nuvem veio mitigar de forma significativa algumas destas limitações, nomeadamente no que diz respeito aos limites de escalabilidade e de tolerância a falhas devido à maior facilidade em recorrer a técnicas de replicação e de elasticidade (*i.e*, a capacidade de aumentar o número de servidores que suportam uma aplicação de acordo com a carga, utilizando técnicas de particionamento). No entanto, notamos que o recurso a infraestruturas na nuvem incorre em custos monetários para os operadores das aplicações que são indesejáveis para estes, ou até mesmo no caso de pequenas e médias empresas que se tentam lançar no mercado já de si bastante competitivo, incomportáveis.

Neste trabalho, quebramos com o modelo comum para o desenho de aplicações web, e propomos uma técnica complementar ao uso de infraestruturas na nuvem para ultrapassar as limitações do modelo cliente-servidor no desenho destas aplicações, com impacto direto sobre a escalabilidade e disponibilidade destes sistemas, a latência sentida por utilizadores geograficamente próximos, e ainda o custo operacional das componentes centralizadas. A ideia principal que guia o nosso trabalho parte da observação de que recentemente foram introduzidas um conjunto de novas tecnologias nos browsers que permitem a comunicação directa e transparente entre os mesmos, seguindo uma metodologia entre pares (do inglês *peer-to-peer* [17,12]) a que no contexto deste trabalho chamamos comunicação *browser-a-browser*. Estas tecnologias incluem o WebRTC [18] (Web Real Time Communication), e a proposta de protocolos de suporte ao contorno de firewall e NATs como o STUN [7] e TURN [8]. Estas novas tecnologias combinadas com novos avanços propostos no HTML5, abrem as portas ao desenvolvimento de frameworks, que permitem o enriquecimento de aplicações web com comunicação direta entre os dispositivos dos utilizadores.

Assim, para ultrapassar as limitações das arquiteturas centralizadas, neste artigo propomos enriquecer as arquiteturas atuais com suporte para comunicação direta browser-a-browser usando as novas tecnologias disponíveis nos browsers. Para tal, apresentamos o desenho de um sistema que permite a aplicações a executarem em múltiplos browsers manterem réplicas de um conjunto de objetos, os quais podem modificar concorrentemente. Este sistema é composto pelos seguintes componentes principais: i) mecanismos para a gestão de filiação que permite a instâncias de uma aplicação web, de acordo com uma política específica a essa aplicação, estabelecer ligações diretas entre os browsers dos utilizadores que interagem mais frequentemente; ii) um conjunto de primitivas de comunicação descentralizadas que permitem às aplicações trocarem informação entre si, e coordenarem-se por forma a propagar a informação para a componente centralizada para alcançar durabilidade, e permitir o acesso aos dados por utilizadores que façam uso de browsers sem suporte para WebRTC; iii) uma biblioteca de CRDTs [16,14] escrita em javascript para simplificar o desenho de aplicações web descentralizadas que executam no browser garantido que o estado da aplicação converge eventualmente.

O sistema que apresentamos neste artigo expõe para as aplicações uma API semelhante ao Google Drive Realtime, a qual é usada para a criação de aplicações colaborati-

vas na Web nas quais um conjunto de utilizadores colaboram partilhando um conjunto de objetos que podem ser atualizados concorrentemente. Além desta solução permitir que se possam usar aplicações já existentes de forma mais simples, permitiu-nos ainda comparar a solução proposta neste artigo com uma solução que apenas usa a componente centralizada. Os resultados obtidos mostram que a solução proposta neste artigo apresenta uma latência bastante inferior.

O artigo está organizado da seguinte forma: a secção 2 apresenta uma visão geral da nossa proposta de arquitetura; na secção 3 discutimos com maior detalhe os componentes principais da arquitetura proposta e interação entre os mesmos; a secção 4 discute detalhes de implementação do nosso protótipo; a secção 5 apresenta a nossa avaliação experimental e uma discussão dos resultados; o trabalho relacionado é abordado na secção 6 e finalmente, a secção 7 conclui o artigo e discute brevemente direções futuras para o trabalho apresentado.

2 Visão Geral

Nesta secção apresentamos uma perspetiva geral sobre a arquitetura proposta para enriquecer a operação de aplicações web colaborativas com comunicação direta browser-a-browser. Note-se que nesta arquitetura a componente centralizada continua a ser utilizada, nomeadamente para garantir a persistência dos dados da aplicação, mas também para suportar clientes que executem a aplicação num browser que não suporte comunicação direta entre browsers, ou para clientes que devido a políticas restritivas de firewalls ou Network Address Translations (NAT), sejam incapazes de receber ligações de outros clientes. Adicionalmente, e como ficará mais claro no resto do artigo, a componente centralizada é também ela alavancada para permitir aos clientes estabelecerem as ligações entre si quando se juntam ao sistema. No entanto começamos por discutir o modelo de interação considerado neste trabalho.

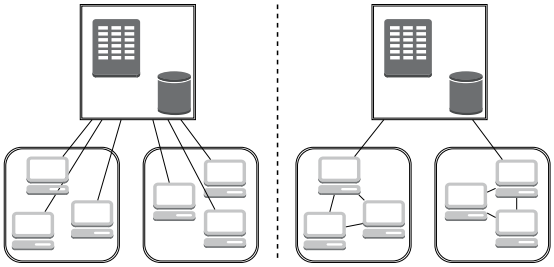


Figura 1: Modelos de comunicação (cliente-servidor) | (browser-a-browser)

2.1 Modelo de Interação

A Figura 1 captura um paralelo entre o modelo de interação comum das aplicações web (esquerda) e o modelo que propomos neste artigo (direita). Como referido anteriormente,

no modelo clássico todas as interações entre clientes são mediadas através de uma componente centralizada, que é também responsável por servir todos os objetos de dados da aplicação e processar as operações que mudem o estado desses objetos. Na nossa proposta estendemos este modelo tradicional oferecendo a hipótese dos clientes (principalmente aqueles que se encontram próximos, e.g, numa mesma rede local) servirem réplicas dos objetos partilhados diretamente, e também de operarem sobre esses objetos localmente, propagando entre eles as atualizações. Esta aproximação permite reduzir substancialmente a dependência da componente centralizada.

Em maior detalhe, o modelo de interação que consideramos neste trabalho é o seguinte. Em primeiro lugar um utilizador recorre ao seu browser para aceder a uma aplicação web, a qual está acessível num servidor web. Ao obter a aplicação web, o browser do utilizador obtém também, de forma transparente, a nossa framework, que é distribuída como código JavaScript, e que é incluído na aplicação web. Após carregar estes recursos, o browser do cliente gera um evento (*onLoad*), que é utilizado pela aplicação para inicializar a nossa framework.

Ao ser inicializada, a nossa framework estabelece uma ligação através de uma WebSocket [9] para um processo servidor, que designamos de *B2BServ*. Este servidor, escrito em NodeJS, é responsável por auxiliar os vários clientes a estabelecerem ligações entre si. A lógica que determina se dois clientes devem ou não estabelecer uma ligação direta browser-a-browser é dependente da aplicação. No entanto, e como discutimos mais tarde, imaginamos um conjunto reduzido de políticas que podem ser benéficas a vários tipos de aplicações. O servidor ao receber o registo de um novo cliente, propaga o pedido de ligação a clientes que já estão ligados aos servidor, mediando assim a ligação inicial entre os clientes.

O novo cliente é então responsável, utilizando o servidor como mediador, por tentar estabelecer as ligações diretas com os vizinhos. Para este fim recorremos aos mecanismos oferecidos pelo Web Real Time Communication (WebRTC) [18], que por sua vez pode recorrer a servidores STUN e TURN como suporte para ultrapassarem firewalls e NATs que possam dificultar o estabelecimento de ligações. Desde o momento da ligação inicial ao servidor a aplicação pode começar a solicitar à framework que esta obtenha cópias de recursos associados à aplicação – por exemplo, no caso de uma aplicação de chat, um recurso pode ser uma lista de mensagens trocadas em cada sala de chat a que o utilizador queira aceder; alternativamente no contexto de uma aplicação de edição colaborativa um recurso pode ser o conteúdo atual de um documento partilhado. Quando um cliente consegue efetuar ligações a outros clientes a comunicação é redirecionada, de forma transparente ao utilizador, para um modelo de interação direta entre clientes.

Os dados são partilhados e replicados entre os clientes usando os CRDTs apropriados para esse tipo de dados. Os CRDTs permitem minimizar a coordenação entre os clientes quando executam operações sobre as suas réplicas locais. Para além disso, permitem garantir de forma simples para o programador, que os objetos replicados entre os clientes convergem para um estado final comum, e também que os clientes têm a oportunidade de agregar várias operações de vários vizinhos numa única mensagem para ser propagada para o repositório central, criando a oportunidade de minimizar a carga imposta sobre esta componente.

Desta forma conseguimos reduzir a latência entre os clientes que estão próximos e reduzir a carga no servidor quando grupos de clientes colaboram entre si e agregam

pedidos. Também torna-se possível evitar o ponto central de falha. Nas arquitecturas actuais, o servidor é o único ponto de comunicação e quando este fica indisponível a interação para. Quando deixamos os clientes interagir diretamente uns com os outros, a interação pode continuar mesmo com falhas temporárias do servidor.

De seguida, e dado este modelo de interação, explicamos em maior detalhe a arquitetura proposta e as responsabilidades de cada uma das suas componentes.

2.2 Arquitectura

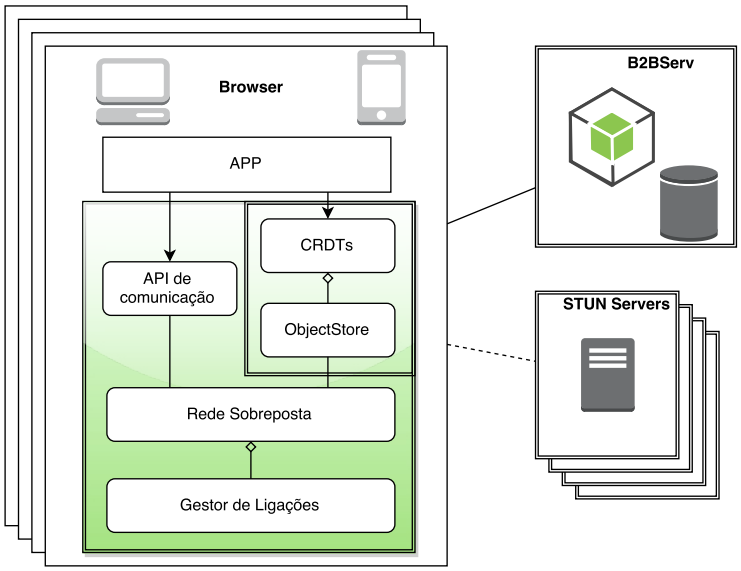


Figura 2: Arquitectura Proposta

Dada a visão geral de interação descrita acima, a Figura 2 esquematiza a arquitetura do sistema que propomos neste artigo, a qual pode ser dividida em dois grandes subcomponentes: o repositório de objetos e o sub-sistema de comunicações. O repositório de objetos é responsável por gerir os objetos a que a aplicação acede, utilizando o subsistema de comunicação para propagar e receber as atualizações efetuadas. O subsistema de comunicações é responsável por efetuar todas as comunicações com outros browsers e com a componente centralizada do nosso sistema.

Uma aplicação a executar no browser, tipicamente escrita em JavaScript, é responsável por interagir com a camada de apresentação (a página web) e por fazer uso da API oferecida pela nossa framework. Esta API permite comunicar diretamente com outros clientes e manter réplicas sincronizadas de um conjunto de objetos partilhados. Para tal, a aplicação deve inicializar a nossa framework quando esta inicia.

Subsistema de comunicação: O subsistema de comunicação é composto pelos seguintes módulos a executarem no cliente:

API de Comunicação: Oferece um conjunto de primitivas de comunicação utilizadas para comunicar com outros clientes (recorrendo à informação mantida pelo módulo *Redes Sobreposta*). A comunicação é feita através de trocas de mensagens codificadas no formato JSON.

Módulo de Gestão da Rede Sobreposta (Rede Sobreposta): Esta componente mantém informação sobre redes sobrepostas usadas para propagar informação entre clientes. Para tal, usa-se a abstração de *grupos* de clientes, em que cada grupo tem associado um conjunto de objetos replicados. Esta componente mantém informação sobre os vizinhos lógicos (i.e, outros clientes) do cliente para cada grupo e implementa algoritmos de disseminação eficientes de mensagens para um grupo, recorrendo às ligações geridas pelo Gestor de Ligações. As operações de disseminação de mensagens para um grupo são utilizadas para garantir a propagação eventual de operações realizadas sobre os objetos replicados em cada grupo.

Gestor de Ligações: Esta componente é responsável por criar e gerir as ligações entre os vários clientes e também com a componente centralizada. A gestão da criação de ligações por esta componente permite evitar o uso de ligações redundantes quer entre clientes quer com a componente centralizada. Para manter ligações entre clientes, esta componente recorre à API oferecida pelo WebRTC. Para manter ligações à componente centralizada usam-se WebSockets. Esta componente colabora ativamente com o módulo de gestão da rede sobreposta por forma a permitir a um cliente juntar-se a um ou vários grupos, de acordo com as necessidade da aplicação e das ações do utilizador.

Para suportar os serviços disponibilizados nos clientes e permitir a comunicação browser-a-browser, o nosso sistema inclui os seguintes módulos a executar na componentes centralizada, potencialmente num ambiente de computação em nuvem:

Componente Centralizada (B2BServ): A componente centralizada é materializada através de um processo escrito em NodeJS. Este servidor recebe ligações produzidas pelos módulos de gestão dos vários clientes (através de WebSockets) e atua como um ponto de entrada dos vários clientes à nossa framework. Em particular esta componente atua como mediador durante o estabelecimento de ligações diretas entre os clientes. Para além disso esta componente é responsável por mediar o acesso dos clientes à camada de persistência centralizada do sistema, isto inclui providenciar o acesso a objetos que não se encontram ainda disponíveis nos clientes, e receber atualizações desses objetos para garantir o seu correto armazenamento na camada (centralizada) de persistência. Esta componente é também responsável por servir os clientes que não tem a capacidade de estabelecer ligações browser-a-browser com outros clientes.

Servidores STUN: Estes servidores encontram-se fora do controle da nossa solução mas são usados para permitir a clientes que se encontrem atrás de firewalls ou NATs o estabelecimento de ligações browser-a-browser diretas. Apesar de que o

operador da aplicação pode correr os seus próprios servidores STUN, no contexto deste trabalho recorremos aos servidores publicamente disponíveis da Google[1].

Repositório de objetos: As aplicações a executarem em diferentes browsers partilham estado recorrendo a um repositório de objetos partilhados. Este repositório de objetos, presente em cada browser, é composto pelos seguintes módulos principais:

Biblioteca de CRDTs (CRDTs): Este módulo oferece um conjunto de tipos de dados baseados nos princípios dos CRDTs codificados em JavaScript. Esta biblioteca é usada simultaneamente pela camada da aplicação para aceder aos dados relevantes da aplicação, e também pelo módulo de gestão de réplicas que materializa as réplicas locais sob a forma de CRDTs. Esta biblioteca pode ser estendida pelo próprio programador por forma a definir novos tipos de CRDTs (por exemplo, que combinem vários dos tipos oferecidos pela framework) que melhor se ajustem às necessidades das suas aplicações.

Módulo de gestão de réplicas: Este módulo é responsável por utilizar os serviços disponibilizados pelo subsistema de comunicação para obter o estado inicial dos objetos e propagar e receber as atualizações efetuadas nos objetos. Adicionalmente, utiliza mecanismos de armazenamento persistente do HTML5 para armazenar de forma persistente as cópias dos CRDT.

3 Suporte B2B

Nesta secção detalham-se os dois subsistemas que compõem a nossa solução: o repositório de objetos e o subsistema de comunicações.

3.1 Repositório de objetos

O repositório de objetos permite, a uma aplicação a executar num browser, a partilha de objetos com outras aplicações a executarem remotamente também em browsers - tipicamente, instâncias da mesma aplicação. Estes objetos partilhados encapsulam os dados manipulados pela aplicação, permitindo aos utilizadores cooperar através da modificação concorrente desses mesmos objetos.

Biblioteca de CRDTs A versão atual do nosso sistema permite partilhar os mesmos objetos que estão disponíveis no Google Drive Realtime [6]: String, Lista e Mapa. Uma *String* mantém uma sequência de caracteres, a qual pode ser modificada através da inserção ou remoção de caracteres. Uma *Lista* mantém uma sequência de valores primitivos ou objetos JavaScript constantes e (referências para) objetos partilhados (Lista, String ou Mapa). Um *Mapa* mantém uma relação entre uma chave, representada por uma string constante, e um valor primitivo ou objeto JavaScript constantes ou (uma

[1] Existe também a hipótese de recorrer a servidores TURN, que reencaminham tráfego entre nós quando, devido a configurações locais de Firewall ou NAT, estes não conseguem estabelecer ligações diretas entre si. No entanto no contexto deste trabalho optámos por não fazer uso destes servidores, o que implica que clientes que não consigam estabelecer ligações entre si, recorrem unicamente aos mecanismos disponibilizados pela componente centralizada.

referências para) um objeto partilhado (Lista, String ou Mapa). Na implementação destes objetos procurou-se fornecer a mesma interação para as aplicações que é fornecida pelo sistema Google Drive Realtime, de forma a simplificar o processo de conversão deste tipo de aplicação para o uso da solução proposta neste artigo.

Ao contrário da solução usada pelo Google Drive Realtime, que recorre a um algoritmo de transformação de operações com servidor central [13], a solução proposta neste artigo usa CRDTs baseados na propagação do estado [16]. Esta solução permite que quaisquer dois clientes possam sincronizar diretamente o estado das suas réplicas em qualquer momento. Assim, os clientes deixam de estar dependentes da componente centralizada e podem, num passo de sincronização, propagar um estado que reflete um conjunto de modificações.

Módulo de gestão de réplicas O módulo de gestão de réplicas tem duas responsabilidades: gerir a persistência dos objetos partilhados usados pelas aplicações e manter esses mesmos objetos atualizados. Para manter os objetos de forma persistente entre sucessivas utilizações da mesma aplicação Web, o sistema recorre aos mecanismos de persistência disponíveis no HTML 5, que permitem a persistência local de estado (ainda que o browser seja terminado).

Para manter os objetos atualizados, o sistema recorre às primitivas de comunicação disponibilizadas pelo subsistema de comunição. Assim, em cada grupo podem-se manter vários objectos partilhados onde a rede sobreposta é usada para propagar as novas versões dos objectos. Na versão atual, usa-se uma solução de propagação lenta de novas versões para permitir a acumulação de múltiplas operações de escrita a cada passo de propagação entre clientes. Essas escritas são encapsuladas numa nova versão do objecto, que é propagada para os elementos do grupo de forma colaborativa como abordado de seguida.

3.2 Subsistema de comunicação

Descrevemos agora alguns detalhes relativos ao desenho das componentes da nossa framework que lidam com os aspetos relativos à comunicação browser-a-browser, e discutimos também o papel da componente centralizada na gestão destas ligações.

Módulo de Gestão das ligações A comunicação direta browser-a-browser é alcançada através do uso da WebRTC API. O WebRTC tem suporte nativo nos browsers mais usados atualmente, nomeadamente o Chrome e o Firefox, sendo possível recorrer a plugins para suportar WebRTC noutros browsers. Um dos motivos principais que motiva o uso de WebRTC é exatamente o suporte nativo nestes browsers, visto que isso permite o uso da nossa framework sem qualquer interação ou esforço (i.e, instalar software ou configurar firewall e NAT) por parte do utilizador final. O WebRTC recorre sempre a conexões seguras, em que todos os dados transmitidos entre dois browsers são cifrados, minimizando assim a exposição dos utilizadores a violações de privacidade.

A existência de Firewalls e NATs sempre constituíram um problema no uso de sistemas entre pares no passado, uma vez que dificultam (ou impedem) a criação de ligações e comunicação direta entre nós que se encontram atrás de um destes componentes. Para contornar este problema, o WebRTC recorre a serviços especialmente desenhados para facilitarem o estabelecimento de ligações nestes cenários. Existem dois tipos de

serviços que podem ser usados. O STUN (que é reflexivo) e permite a ambos os clientes determinarem os seus IPs públicos (no caso de NAT) e instalar estado nas firewalls ou NATs para permitir comunicação direta. A segunda alternativa passa pelo uso do serviço TURN, em que os servidores que providenciam este serviço apenas redirecionam o tráfego entre os dois nós.

No trabalho apresentado neste artigo, o uso do serviço TURN foi inibido, visto que nos cenários em que nos focamos, é mais eficiente para os clientes comunicarem indiretamente através da componente centralizada do sistema do que terem toda a sua comunicação direta redirecionada para um servidor que a aplicação não controla. Nestes casos, assim como no caso em que o browser não suporta WebRTC, a framework utiliza o modelo de interação típico das aplicações Web, em que todas as ações dos utilizadores são mediadas pela componente centralizada.

De forma a estabelecer uma ligação direta browser-a-browser, o WebRTC estabelece um protocolo de *Signalling* que requer a ambos os clientes a troca de informação inicial entre si sob a forma de *offers* (em formato textual). Qualquer mecanismos que permita a troca de texto pode suportar a execução do protocolo de *Signalling*. Na nossa framework, optámos por recorrer à componente centralizada, utilizando WebSockets, para efetuar a troca de informação necessária ao estabelecimento destas ligações. Esta decisão prende-se com o facto de que a maioria de serviços Web requer um processo de autenticação por parte do utilizador, o que requer necessariamente o envio de informação para a componente centralizada no início da sessão.

Componente da Rede Sobreposta Como discutido anteriormente, os vários clientes da aplicação organizam-se no contexto de grupos. Para tal a componente de filiação da nossa framework oferece uma operação de Join que requer o identificador desse grupo. Para permitir a um cliente juntar-se a um grupo com identificador $\mathcal{G}$, a nossa framework envia um pedido à componente centralizada (através da ligação WebSocket) com esse identificador e solicita a filiação corrente do grupo. Este pedido carrega em *piggyback* a mensagem desse cliente para a execução do protocolo de *Signalling* do WebRTC.

Ao receber este pedido a componente centralizada, que mantém informação (parcial) relativa à filiação dos grupos, envia ao cliente o subconjunto de membros do grupo $\mathcal{G}$, e notifica também os restantes elementos do grupo sobre a chegada do novo cliente. As mensagens enviadas para cada cliente pela componente centralizada serve também para enviar a informação (originalmente enviada por cada cliente) para a execução do protocolo de *Signalling*. Ao receberem estas notificações, cada cliente recorre então à componente de gestão da nossa framework para materializar as novas ligações.

O resultado final deste processo é que os clientes, para cada grupo, estabelecem uma rede-sobreposta similar a um grafo aleatório e com ligações bi-direcionais, com propriedades similares às redes sobrepostas mantidas por algoritmos utilizados no contexto dos sistemas entre pares como o Cyclon [17] ou o HyParView [12].

Para evitar sobrecarregar a componente centralizada com atualizações excessivas relativas às operações realizadas pelos clientes sobre os objetos que replicam localmente, apenas um pequeno subconjunto dos clientes mantém a ligação à componente centralizada. Para decidir que clientes são responsáveis por esta tarefa recorremos a um algoritmo de Bully para eleição de líderes locais [2]. Neste algoritmo cada cliente começa num estado inicial *ativo* e periodicamente emite para todos os seus vizinhos uma mensagem contendo o seu identificador. Sempre que um cliente recebe uma mensagem

de outro cliente cujo identificador seja menor que o seu, muda o seu estado para *inerte*, pára de transmitir esta mensagem periódica e termina a sua ligação com a componente centralizada. Um nó *inerte* que não receba mensagens destas por um período suficientemente grande, muda o seu estado para *ativo* e restabelece a ligação para a componente centralizada. Apenas os nós ativos são responsáveis por interagir com a componente centralizada.

Esta componente é também responsável por suportar os mecanismos de comunicação disponibilizados pela API de comunicação. Para tal existem mecanismos para comunicação ponto-a-ponto, e também mecanismos de difusão para um grupo. Estes últimos recorrem a mecanismos de difusão epidémica bem conhecidos para propagaram informação entre todos os elementos do grupo [1].

API de comunicação A API de comunicação oferece mecanismos que permitem à aplicação (e às restantes componentes da framework) comunicarem com outros clientes no contexto de um grupo. Esta API oferece mecanismos para o envio de mensagens ponto-a-ponto e multicast, sempre no contexto de um grupo a que o cliente pertença. As mensagens são sempre propagadas usando uma política de melhor esforço.

Componente Centralizada Como discutido anteriormente, a componente centralizada é materializada por um servidor aplicacional escrito em NodeJS. Este é responsável por manter informação sobre os grupos de comunicação ativos em cada momento, conhecendo em cada grupo um pequeno conjunto de elementos que se encontrem ativos (os nós que se mantém num estado *ativo* após a execução do algoritmo de *bully* ao nível da componente de filiação).

Esta componente é também responsável por receber as alterações efetuadas pelos clientes sobre os dados da aplicação e garantir a persistência dessas alterações na componente centralizada. Para além disso o servidor serve de ponto de contacto para facilitar o estabelecimento de ligações browser-a-browser e opera como ponto de acesso à aplicação para os clientes que não consigam estabelecer ligações diretas browser-a-browser.

4 Detalhes de Implementação

De forma a demonstrar a exequibilidade da nossa proposta desenvolvemos um protótipo que usaremos de seguida no nosso trabalho experimental. O nosso protótipo, incluindo código da componente centralizada, foi implementado em 3.863 linhas de código JavaScript. O código JavaScript utilizado para materializar a componente da framework que executa no cliente, quando compactado de forma automática[2], fica com um tamanho total de 39 KB.

As páginas web e o código JavaScript das aplicações e da framework pode ser servidas por qualquer servidor Web. A autenticação perante o serviço Web é da responsabilidade do programador. Nas nossas experiências efetuamos a autenticação ao nível do B2BServ na componente centralizada.

A nossa implementação do B2BServ suporta particionamento desta componente em vários processos para garantir a escalabilidade do sistema para grandes números de

[2] Recorrendo à ferramenta disponível em `https://github.com/mishoo/UglifyJS2`.

clientes e/ou objetos a serem acedidos. A lógica de particionamento é guiada através dos identificadores de grupos.

Simplificamos a implementação do nosso componente de gestão de filiação, e ao invés de recorrer a um grafo aleatório para manter as ligações entre os vários clientes de cada grupo, mantemos todos os clientes de um grupo ligados entre si num grafo totalmente conexo. Esta simplificação foi feita para minimizar o tempo de desenvolvimento do protótipo e será endereçada em trabalho futuro.

O módulo de gestão de objetos foi desenhado especificamente para lidar com objetos do tipo CRDT. Foram implementadas funções para obter, criar e apagar objetos. A implementação desta componente assume que todos os objetos devem ter definidas funções *compare* e *merge*, que são típicas às interfaces de CRDTs. A biblioteca de CRDTs fornecida atualmente com a framework providencia implementações de Strings, Listas e Mapas descritas em [16].

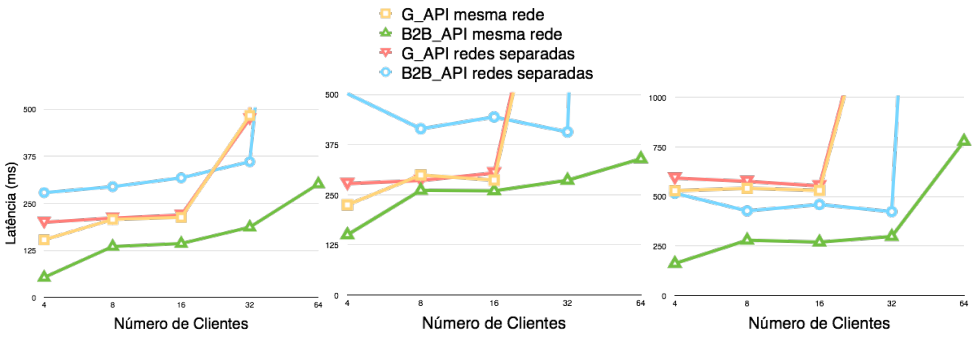
A camada de rede sobreposta suporta dois mecanismos de disseminação, um primeiro baseado em inundação em que todos os clientes enviam as mensagens para todos os outros clientes, e um segundo modo em que os clientes trocam informação através dos nós cujo estado permanece como *ativo* devido à execução do protocolo de *bully* anteriormente descrito. Este último protocolo pode ser parametrizado com o tempo entre transmissões dos identificadores dos nós em estado *ativo*. A sincronização do estado das réplicas dos clientes com a componente centralizada é também configurável, sendo tipicamente de vários segundos para permitir aos clientes responsáveis por esta operação a possibilidade de agregarem operações de vários clientes em cada passo de sincronização com o servidor.

De forma a testar o protótipo do sistema foram desenvolvidas várias aplicações de demonstração. Estas aplicações permitiram também aferir a facilidade do uso da nossa framework. Sem contabilizar o número de linhas de código para implementar a interação com o ambiente web (i.e, JavaScript para manipular o conteúdo da página HTML) as aplicações desenvolvidas incluem os seguintes exemplos: uma aplicação de chat com salas, que foi escrita em quatro linhas de código; uma lista que pode ser editada colaborativamente, a qual foi desenvolvida em seis linhas de código, entre outros exemplos. Omitimos os detalhes relativos às aplicações de exemplo devido a limites de espaço.

5 Avaliação

Para avaliar o nosso sistema comparamos o seu desempenho com o desempenho obtido utilizando uma aplicação que recorre ao Google Realtime API. Relembramos que a Google Realtime API (G_{API}) suporta os mesmos objetos, os quais podem ser partilhados por múltiplos clientes. Todos os nossos testes foram realizados utilizando instâncias *m3.xlarge* da Amazon Web Services EC2.

Em todos os testes colocámos um único servidor na zona us-west-1 (Califórnia) e dividimos os clientes de forma igual entre 16 máquinas, 8 na zona us-west-2 (Oregon) e 8 na zona us-east-1 (Virgínia). Desta forma dividimos os clientes em dois grupos de igual tamanho (cujo tamanho variamos nas nossas experiências), e apenas permitimos ligações diretas browser-a-browser entre máquinas localizadas na mesma região (i.e,



(a) Média (b) Percentual 95 (c) Máxima

Figura 3: Latência entre clientes.

na mesma rede local). Esta restrição não existe na nossa framework, no entanto esta configuração promove ligações entre clientes de baixa latência.

Em todas as experiência recorremos ao servidor na Califórnia para atuar como servidor Web e executar o processo B2BServ. Todos os clientes executam o cliente no Chromium, a base do Google Chrome de código aberto, em modo headless que registas as suas actividades através do Xvfb, um servidor de écran virtual, em memória.

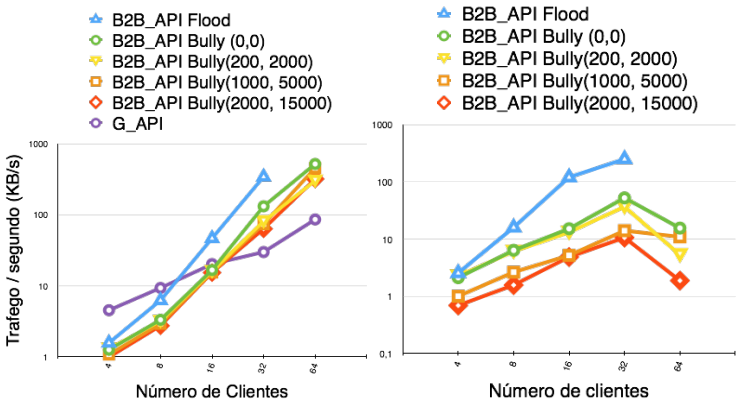
Os objetivos principais do nosso trabalho experimental são os seguintes: i) avaliar a latência observada pelos vários clientes em ambos os sistemas (secção 5.1); ii) medir a quantidade de dados trocados entre o servidor e os clientes em ambas as soluções e entre os clientes na nossa solução (secção 5.2); e iii) avaliar o suporte da nossa framework à operação desconetada do servidor (secção 5.3).

5.1 Latência entre clientes

Neste teste usamos um cliente que efetua escritas (localizado na Virgínia) e medimos o tempo até as escritas serem propagadas a todos os outros clientes. O cliente que efetua escritas escreve um carácter numa *string* em cada uma das suas operações.

Apresentamos os resultados de latência entre clientes numa mesma rede local (i.e, que recorrem a comunicação direta browser-a-browser) e clientes em redes separadas (i.e, atualizações têm que passar pela componente centralizada). Para facilitar a compreensão destes resultados indicamos de seguida os valores de latência entre as várias máquinas usadas neste teste, medida com a ferramenta `ping`: i) entre Oregon e o servidor na Califórnia: 20 ms; ii) entre Virgínia e o servidor na Califórnia: 80 ms; iii) entre Oregon e o servidor da Google: 13 ms; iv) entre Virgínia e o servidor da Google: 12 ms; e v) entre Oregon e Virgínia: 70 ms.

A Figura 3(a) apresenta a latência média, que se apresenta, como esperado, substancialmente mais baixa no caso em que os clientes comunicam diretamente através de ligações browser-a-browser. As figuras 3(b) e 3(c) apresentam os valores de latência para o percentil 95 e o valor máximo respetivamente. Estes resultados corroboram os resultados anteriores e mostram que no caso da comunicação mediada pela componente



(a) Trafego no servidor (b) Trafego por cliente (média)

Figura 4: Trafego

centralizada, os valores de latência são substancialmente mais elevados. Mais ainda, estes resultados mostram que a latência sentida pelos clientes quando recorrem à nossa framework não sofre variações significativas com o aumento de número de clientes. Os resultados mostram igualmente que o sistema baseado na Google Realtime API é muito sensível ao número de clientes, sendo que a latência aumenta substancialmente com o número total de clientes.

5.2 Tráfego

Para medir o tráfego gerado por cada alternativa, efetuamos duas escritas por segundo em todos os clientes durante um curto período de tempo, variando o número de clientes. Para além disso variamos também o protocolo de disseminação de mensagens entre os clientes recorrendo à disseminação por inundação (*Flood*) e recorrendo a comunicação baseada nos nós ativos após a execução do algoritmo de *Bully*. Neste último caso variamos também o tempo de propagação de mensagens dos nós ativos para a componente centralizada.

A Figura 4(a) apresenta a média do tráfego acumulado por segundo no servidor. Os resultados mostram que a solução de disseminação baseada em inundação leva o sistema a ficar facilmente sobrecarregada, tal não acontece com o mecanismo de disseminação alternativo, que, tal como esperado, mostram uma menor carga imposta sobre o servidor conforme o tempo de propagação de atualizações para o servidor aumenta. O leitor deve notar que estes resultados mostram o acumulado obtido na totalidade da experiência que inclui a transmissão de dados necessária para descarregar o código HTML e JavaScript do servidor assim como o estabelecimento de ligações entre os clientes. Visto que a duração da experiência é relativamente baixa, estes custos seriam amortizados em execuções mais longas.

Na Figura 4(b) apresentamos o tráfego médio por segundo em cada cliente. É notório que este tráfego varia com o número de clientes, ou seja, quando o número

de utilizadores no sistema aumenta, aumenta também a quantidade de tráfego gerado. Isto contece devido a um aumento do número de ligações a fazer entre os clientes e a interação directa entre os mesmos.

5.3 Suporte a desconexão e outros ambientes de execução

Suporte a desconexão: Efetuamos experiências onde desligámos o acesso à componente centralizada durante a execução das experiências onde os vários clientes efetuam escritas, ou seja, desconexão física à rede exterior (Internet). Esta experiência revelou que os clientes que tinham ligações diretas entre si continuavam a cooperar ativamente. Estes resultados (cujos detalhes omitimos devido a restrições de espaço) validam este aspeto do desenho da nossa framework.

Outros dispositivos: Verificámos a capacidade de usar a nossa framework num conjunto variado de dispositivos, que incluíram portáteis, telemóveis, e tablets ligados através de uma rede local. Verificámos que estes dispositivos conseguiam criar ligações entre si, permitindo a sua comunicação sem necessidade de contactar a componente centralizada.

6 Trabalho Relacionado

Serviços colaborativos baseados em Web tipicamente armazenam os dados dos clientes em servidores, os quais podem ser replicados geograficamente para garantir uma mais baixa latência e elevada disponibilidade. O uso de consistência forte para este tipo de aplicações tende a não ser adequado devido a impor restrições ou atrasos à execução de operações pelos utilizadores. Assim, estes sistemas usam tipicamente soluções de consistência eventual (ou causal), as quais incluem técnicas de resolução de conflitos para unificar as modificações executadas concorrentemente quando estes (inevitavelmente) aparecem.

Vários sistemas suportam a edição colaborativa entre clientes em tempo real recorrendo a solução de transformação de operações (OT) [13]. Nestas soluções, as operações são transformadas antes de serem executadas na réplica de cada cliente. Apesar de terem sido propostos algoritmos em que esta transformação ocorre de forma distribuída em cada cliente, os sistemas normalmente usam algoritmos que recorrem a uma componente centralizada que efetua a transformação das operações. O Etherpad [3] e o ShareJS [5] são sistemas de processamento de texto baseado na Web que usam servidores próprios que transformam as operações dos clientes. A Google Realtime API[6] requer o uso dos servidores e autenticação da própria Google, mas tem suporte a mais tipos de dados como Listas e Mapas.

As soluções de OT foram extensivamente estudadas na literatura, especialmente pela comunidade de edição colaborativa, e muitos algoritmos de OT foram apresentados. Contudo foi demonstrado que a maioria dos algoritmos de OT para sistemas descentralizados são incorretos [10]. Os CRDTs, Convergent (ou Commutative) Replicated Data Types, surgiram inicialmente no contexto da edição colaborativa [14], como uma aproximação alternativa ao OT que permite uma solução de sincronização entre pares ao mesmo tempo simples e correta. Os CRDTs [16] são tipos de dados replicados que

garantem a convergência por desenho, permitindo a execução imediata das operações num cliente sem necessidade de coordenação.

Os CRDTs têm encontrado sucesso em várias aplicações distribuídas, por exemplo, a base de dados NoSQL Riak[11] é um exemplo de uma key-value store distribuída com alta disponibilidade que recorre internamente a CRDTs. O SwiftCloud [15] é outro sistema de armazenamento de dados eventualmente consistente que usa CRDTs para manter caches do lado do cliente. O nosso sistema difere destes sistemas ao permitir aos clientes sincronizarem diretamente as réplicas dos objetos partilhados entre os clientes.

A solução que propomos neste artigo recorre à comunicação direta entre os clientes, seguindo um modelo entre-pares (do inglês *peer-to-peer*). Os sistemas entre-pares foram já amplamente estudados, sendo que existem soluções propostas na literatura para a gestão descentralizada de redes sobrepostas [12,17,4] e também para a disseminação eficiente e tolerante a faltas de informação [12,1]. Neste trabalho inspiramo-nos em algumas destas soluções para construir o subsistema de comunicação, em particular o módulo de gestão de redes sobreposta.

Um trabalho recente, Priv.io [19] recorre ao uso de WebRTC para suportar uma rede social com aspetos descentralizados. Este sistema pretende resolver problemas de privacidade nas redes sociais atuais, garantindo que toda a componente centralizada destes sistemas seja garantida por um conjunto de recursos federados (na nuvem) controlados (e pagos) pelos utilizadores. Para permitir a interação entre os utilizadores, o Priv.io recorre a uma aplicação JavaScript suportada por WebRTC para permitir aos clientes a troca direta de chaves criptográficas, entre outras interações em tempo real (e.g, conversação). No entanto, o foco principal deste trabalho passa pela privacidade dos dados dos utilizadores, evitando o uso de uma componente centralizada que serve de repositório de dados.

7 Conclusões e Trabalho Futuro

Neste artigo apresentamos uma arquitetura alternativa para o suporte de aplicações Web colaborativas com comunicação direta e transparente browser-a-browser. Implementámos esta arquitetura sob a forma de uma framework que recorre a WebRTC para materializar a comunicação direta entre browsers, e recorre a CRDTs para materializar réplicas locais nos clientes, que podem ser propagas diretamente entre estes, atualizadas localmente e reconciliadas entre os vários clientes, para além de serem propagadas para uma componente centralizada para garantir persistência e compatibilidade com clientes que não suportem WebRTC. Com base nesta framework implementámos um conjunto de aplicações simples, e uma aplicação de edição colaborativa que comparámos experimentalmente com uma aplicação similar sobre a Google Drive Realtime API.

Como trabalho futuro pretendemos melhorar o desenho de várias das componentes da nossa framework, em particular o desenho da componente de rede sobreposta e de comunicação. Pretendemos também estudar no contexto deste sistema o *trade-off* existente entre várias alternativas de desenho de CRDTs, nomeadamente entre as alternativas baseadas em operações e estado.

Agradecimentos: Este trabalho foi parcialmente suportado pelos projectos FCT/MEC NOVA LINCS PEst UID/CEC/04516/2013 e pelo projecto Europeu FP7 SyncFree (grant agreement 609551).

Referências

1. Birman, K.P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., Minsky, Y.: Bimodal multicast. ACM Transactions on Computer Systems (TOCS) 17(2), 41–88 (1999)
2. Coulouris, G.F., Dollimore, J., Kindberg, T.: Distributed systems: concepts and design. pearson education (2005)
3. EtherpadFoundation: Etherpad, http://etherpad.org
4. Ganesh, A.J., Kermarrec, A.M., Massoulié, L.: Scamp: Peer-to-peer lightweight membership service for large-scale group communication. In: Networked Group Communication, pp. 44–55. Springer (2001)
5. Gentle, J.: Sharejs api, https://github.com/share/ShareJS#client-api
6. Google: Realtime api, https://developers.google.com/drive/realtime
7. IETF: Stun, https://tools.ietf.org/html/rfc5389
8. IETF: Turn, https://tools.ietf.org/html/rfc5928
9. IETF: Websocket, https://www.websocket.org
10. Imine, A., Rusinowitch, M., Oster, G., Molli, P.: Formal design and verification of operational transformation algorithms for copies convergence. Theor. Comput. Sci. 351(2), 167–183 (Feb 2006), http://dx.doi.org/10.1016/j.tcs.2005.09.066
11. Klophaus, R.: Riak core: building distributed applications without shared state. In: ACM SIGPLAN Commercial Users of Functional Programming. p. 14. ACM (2010)
12. Leitao, J., Pereira, J., Rodrigues, L.: Hyparview: A membership protocol for reliable gossip-based broadcast. In: Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on. pp. 419–429. IEEE (2007)
13. Nichols, D.A., Curtis, P., Dixon, M., Lamping, J.: High-latency, low-bandwidth windowing in the jupiter collaboration system. In: Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology. pp. 111–120. UIST '95, ACM, New York, NY, USA (1995), http://doi.acm.org/10.1145/215585.215706
14. Preguica, N., Marques, J.M., Shapiro, M., Letia, M.: A commutative replicated data type for cooperative editing. In: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems. pp. 395–403. ICDCS '09, IEEE Computer Society, Washington, DC, USA (2009), http://dx.doi.org/10.1109/ICDCS.2009.20
15. Preguiça, N., Zawirski, M., Bieniusa, A., Duarte, S., Balegas, V., Baquero, C., Shapiro, M.: Swiftcloud: Fault-tolerant geo-replication integrated all the way to the client machine. In: Reliable Distributed Systems Workshops (SRDSW), 2014 IEEE 33rd International Symposium on. pp. 30–33. IEEE (2014)
16. Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M.: Conflict-free replicated data types. In: Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems. pp. 386–400. SSS'11, Springer-Verlag, Berlin, Heidelberg (2011), http://dl.acm.org/citation.cfm?id=2050613.2050642
17. Voulgaris, S., Gavidia, D., Van Steen, M.: Cyclon: Inexpensive membership management for unstructured p2p overlays. Journal of Network and Systems Management 13(2), 197–217 (2005)
18. W3Cs: Webrtc, http://w3c.github.io/webrtc-pc/
19. Zhang, L., Mislove, A.: Building confederated web-based services with priv.io. In: Proceedings of the First ACM Conference on Online Social Networks. pp. 189–200. COSN '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2512938.2512943

Detecção eficiente de comportamento parasita em sistemas de difusão entre-pares.

João Silva, Xavier Vilaça, Luís Rodrigues, Hugo Miranda

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
{joao.roque,xavier.vilaca,ler}@tecnico.ulisboa.pt,
Universidade de Lisboa, Faculdade de Ciências, LASIGE,
hamiranda@ciencias.ulisboa.pt

Resumo Neste artigo estudamos técnicas que permitem evitar comportamentos parasitas em sistemas de difusão entre-pares. Para este efeito, desenvolvemos uma variante de uma rede sobreposta em que todos os nós possuem vistas parciais simétricas, de tamanho semelhante. Estas vistas são usadas para promover interacções frequentes entre vizinhos, que facilitam a monitorização e a classificação dos parceiros de forma localizada e eficiente. Esta classificação é usada para aplicar sistemas de penalização simples, que permitem detectar e excluir os nós parasitas em sistemas de transmissão contínua de dados (streaming). A avaliação mostra que este mecanismo limita a assimetria na utilização de recursos mesmo na presença de comportamentos mais sofisticados.

1 Introdução

A comunicação descentralizada entre-pares é uma técnica interessante para explorar os recursos sub-utilizados que existem na periferia da rede. Infelizmente, existem evidências de que, em sistemas entre-pares, uma fracção dos nós (designados por nós parasitas) pode não contribuir para o sistema, explorando abusivamente o esforço dos restantes os quais cooperam incondicionalmente (designados por nós altruístas) [1, 3, 9]. O comportamento dos nós parasitas impõe uma carga injusta nos nós altruístas e degrada a execução de tarefas. Mecanismos que permitem detectar e limitar o comportamento parasita têm, portanto, grande relevância prática.

Neste artigo abordamos o comportamento parasita na transmissão contínua de dados (streaming) em sistemas entre-pares. As vantagens da difusão entre pares para suportar este tipo de aplicações têm sido confirmadas por diversos exemplos reais de larga escala, incluindo PPLive [8] entre outros [16]. Neste contexto, o trabalho relacionado [7, 14, 15] tem tratado comportamentos parasitas assumindo que todos os nós são racionais. Isto é, os nós não contribuem porque tentam maximizar a sua utilidade, a qual diminui com a quantidade de recursos fornecidos aos restantes nós. Como os nós racionais se conseguem desviar do protocolo de formas arbitrárias, é necessário fornecer incentivos sofisticados, que geralmente acarretam uma penalização no desempenho do sistema ou requerem

algum tipo de controlo centralizado. No entanto, na prática, não é realista esperar que cada nó seja capaz de calcular a melhor estratégia que maximize a sua utilidade. Desta forma, fazemos as seguintes hipóteses que, acreditamos, representam um cenário mais realista. Primeiro, assumimos que uma grande fracção de nós é altruísta. Segundo, assumimos que a maior parte dos nós que se desviam do protocolo são parasitas puros, isto é, não partilham qualquer tipo de recurso. Terceiro, consideramos que os nós podem atacar o sistema através de lavagem de identidade *(white-washing)*, isto é, podem sair e re-entrar no sistema com uma nova identidade. Quarto, admitimos que apenas uma pequena fracção de nós é racional. Com esta configuração, basta assegurar que: *i*) os parasitas são detectados e expulsos de forma eficiente e não obtêm beneficios quando tentam re-entrar no sistema com nova identidade; *ii*) nós racionais precisam de contribuir para o sistema com uma quantidade de recursos razoável para conseguirem receber os conteúdos; *iii*) os nós altruístas têm mecanismos para detectar que as hipóteses sobre o comportamento do sistema foram violadas (por exemplo, no caso onde, contra as expectativas, existe uma grande fracção de nós racionais entre a população) de forma a poderem activar mecanismos de controlo mais robustos (e também com custo mais elevado), mas apenas quando necessário.

Tal como na maioria das concretizações de transmissão contínua de dados, no nosso sistema é necessário que os nós se juntem a uma rede sobreposta para receberem o conteúdo. Em vez de suportar topologias arbitrárias da rede sobreposta, o nosso protocolo incentiva os nós a manterem ligações simétricas com um grupo pequeno de vizinhos. Sendo assim, é possível usar estratégias do tipo "olho-por-olho" para detectar nós parasitas, em vez de mecanismos complexos de controlo e reputação que podem acarretar uma grande carga adicional. Com base nos princípios acima, desenvolvemos o FastRank, um sistema que faz gestão integrada da topologia e um esquema de monitorização dos nós que minimiza eficazmente o impacto de nós parasitas em aplicações de transmissão contínua de dados. O FastRank implementa um esquema de classificação de vizinhos que permite aos nós altruístas ficarem ligados a outros nós altruístas, enquanto os nós parasitas são rapidamente penalizados por não contribuirem para o sistema e não podem evitar esta penalização mudando de identidade.

Mostramos também que a nossa abordagem é robusta a comportamentos mais sofisticados, tais como às tentativas de manipulação da topologia para benefício próprio, ou esforços de contribuição minimalista. Em particular, mostramos que para estes comportamentos terem sucesso, o atacante tem de contribuir com uma quantidade razoável de recursos. Estes resultados são suportados por uma extensa avaliação do FastRank, com diferentes parâmetros de topologia da rede sobreposta e percentagens de nós parasitas.

O resto do artigo está organizado da seguinte forma. A Secção 2 aborda o trabalho relacionado. A Secção 3 descreve a arquitectura e algoritmos do FastRank. Os modelos de comportamentos dos nós são discutidos na Secção 4. A Secção 5 reporta os nossos resultados experimentais. Finalmente, a Secção 6 conclui o artigo.

2 Trabalho Relacionado

Existem duas abordagens principais para suportar a transmissão contínua de dados em sistemas de larga escala: uma é construir árvores de disseminação e a outra consiste em usar protocolos epidémicos. Em condições ideais, i.e. em sistemas estáveis e com um número desprezável de nós parasitas, as abordagens em árvore oferecem uma estrutura eficiente para disseminar a informação, porque evitam qualquer tipo de redundância. No entanto, em ambientes não controlados como a Internet, a qualidade das árvores rapidamente decresce devido à necessidade de reconstrução sempre que um nó se desliga ou demonstra um comportamento parasita.

Um protocolo de disseminação epidémico pode ser decomposto em dois componentes. O componente de filiação é responsável pela definição e manutenção da topologia de rede. Um grande desafio está na definição de uma topologia que seja resiliente, mitigando o impacto das entradas e saídas. Exemplos de algoritmos com aproximações distintas a este problema são o SCAMP [6] e o HyParView [13]. A segunda componente de um protocolo epidémico consiste no processo de disseminação. Tipicamente, os nós trabalham por turnos, e seleccionam aleatoriamente um subconjunto da vista, cuja dimensão é denominada *fanout*, para enviarem a informação. Esta estratégia de difusão é eficaz e resiliente em cenários onde todos os nós cooperam. No entanto, nós racionais, não irão contribuir para a disseminação da informação beneficiando assim da economia de resursos conseguida. Do ponto de vista do protocolo, nós racionais são comparáveis a nós que falham, e por isso o seu impacto pode, até certo nível, ser atenuado pela robustez e redundância que caracterizam os protocolos epidémicos. Ainda assim, é necessário concretizar mecanismos para prevenir que este problema se agrave e resulte num rácio de retransmissões insuficiente.

De forma a mitigar o impacto de participantes racionais, o BAR Gossip [15] utiliza um mecanismo de trocas balanceadas. A informação é cifrada antes de ser enviada e o emissor apenas publica a chave após ter recebido, do mesmo nó, uma quantidade de informação semelhante. O mecanismo de trocas balanceadas pode ser visto como uma materialização da aborgadem "olho-por-olho" que é praticada no BitTorrent [3]. Infelizmente, a cifra adiciona um custo considerável na troca de informação. O FlightPath [14] utiliza duas aproximações para atenuar o custo da cifra imposta pelo BAR Gossip. Por um lado, utiliza um mecanismo de controlo de fluxo, onde os nós negoceiam antecipadamente que pacotes vão trocar. O controlo de fluxo permite balancear o tráfego a cada ronda para diminuir a redundância. Em segundo lugar, rastreia as trocas entre vizinhos, o que substitui o modelo do BAR Gossip por um mecanismo mais eficiente onde os nós têm uma margem que lhes permite não enviar informação. O LiFTinG [7] implementa mecanismos distribuídos para detectar comportamentos racionais em ambientes assimétricos, i.e. quando é esperado que um nó envie mais informação do que aquela que recebe desse mesmo nó. O LiFTinG exige que os nós confirmem o comportamento de outros nós através de validações cruzadas ao histórico das interacções passadas. Infelizmente, o cruzamento de informação pode impor uma sobrecarga no sistema que se aproxima dos 10%.

3 FastRank

Descrevemos agora o FastRank, um serviço de difusão resiliente a comportamentos parasitas. O FastRank tem três componentes principais: construção e manutençao da topologia da rede, um esquema de classificação localizado, e uma estratégia de disseminação. Estes componentes cooperam entre si para evitar que os nós do sistema possam beneficiar de comportamentos parasitas. Em particular, a manutenção da rede estabelece mecanismos que encorajam os nós a manterem um número pequeno de ligações estáveis e bi-direccionais. Isto obriga os nós a interagirem entre si frequentemente. Estas interacções podem ser facilmente monitorizadas localmente, sem ser necessário a disseminação de tráfego de controlo adicional. Desta forma, a monitorização local torna-se muito eficiente, e é usada por nós altruístas para expulsarem da sua vista nós que aparentem ter comportamentos parasitas. Os nós parasitas podem tentar re-estabelecer ligações com outros membros da rede. No entanto, os protocolos de gestão da topologia incluem mecanismos para assegurar que estas operações são morosas, o que prejudica a recepção dos conteúdos pelos nós parasitas.

3.1 Organização da Rede

A ideia principal do FastRank consiste em utilizar uma rede sobreposta estável, com ligações simétricas, para efectuar a disseminação de mensagens. Dois nós altruístas, após estabelecerem uma ligação, irão preservá-la até que um deles falhe. O número de vizinhos de cada nó é deliberadamente pequeno (i.e, logaritmico ao tamanho do sistema) [11] de forma a que os vizinhos tenham de interagir frequentemente durante o processo de disseminação das mensagens, permitindo a detecção rápida e eficiente de nós parasitas.

O HyParView [13] é um protocol que possui características que se aproximam bastante do nosso objectivo. Os autores do HyParView demonstraram experimentalmente que a rede suporta eficazmente a difusão fiável de mensagens [12]. Infelizmente, não é possível utilizar o HyParView no desenvolvimento do FastRank sem alterar alguns dos seus mecanismos. Isto porque o HyParView foi desenvolvido assumindo que todos os nós do sistema são altruístas. Por esta razão, também não possui mecanismos para prevenir a "lavagem de identidades" (do Inglês, *white washing)*, onde um nó estabelece constantemente novas ligações a diferentes nós. Como os nós demoram algum tempo a identificar o comportamento de um novo vizinho, um nó parasita irá receber informação até a ligação ser quebrada. Se um nó parasita consegue estabelecer novas ligações ao mesmo ritmo que perde as antigas, então ele consegue receber todos os conteúdos, mesmo que os seus vizinhos o identifiquem como parasita. Por esta razão, o FastRank implementa uma variante do HyParView. Esta adaptação do HyParView, a que chamamos *HyParView com Restrições*, implementa mecanismos que limitam o ritmo a que um nó consegue estabelecer novas ligações.

HyParView O objectivo do HyParView consiste na criação de uma rede onde cada nó está ligado a um número pequeno de vizinhos que representam uma

amostra aleatória da população total. O conjunto de vizinhos de cada nó é designado por *vista activa*. O HyParView assegura que as vistas parciais são *simétricas*, i.e., se n_1 está na vista parcial de n_2, então, n_2 também está na vista parcial de n_1. A topologia resultante tem várias propriedades interessantes para os nossos objectivos: *i)* existe um limite máximo para o grau incidente de cada nó, e *ii)* se cada nó tiver uma vista parcial completa (*i.e.* se o grau-divergente de todos os nós for igual ao tamanho da vista parcial), então cada nó do sistema tem exactamente o mesmo número de ligações. Para aumentar a robustez da rede, cada nó tem também uma vista *passiva* com os identificadores de nós adicionais que são usados para substituir as ligações perdidas na vista activa (devido a falhas ou disconexões). A vista activa é usada para suportar as aplicações que usam o HyParView, como disseminação de mensagens. Como a vista activa é pequena e baseada em ligações simétricas, os vizinhos interagem frequentemente. No HyParView original, isto é usado para detectar falhas. No FastRank, isto é também usado para detectar comportamentos parasitas.

HyParView com Restrições O HyParView com restrições é uma adaptação do protocolo original HyParView, que foi desenvolvido para resolver os requisitos do FastRank. Esta variante inclui mecanismos que limitam o ritmo a que um nó consegue estabelecer novas ligações. Deste modo, quando um nó i contacta outro nó j, para estabelecer uma nova ligação entre i e j, o nó i recebe uma tarefa morosa que tem de resolver antes do pedido de ligação ser aceite. Na prática, é introduzido um *período de quarentena* antes de uma nova ligação ser estabelecida. Este período de quarentena deverá ser suficientemente longo de forma a que várias tramas sejam perdidas e que a lavagem de identidade deixe de compensar. Além disso, a tarefa dada ao nó i deverá impossibilitar a sua resolução em paralelo por um único nó durante o período de quarentena, i.e., se um nó tenta estabelecer 2 novas ligações, ele deverá ser forçado a sofrer um atraso correspondente a 2 períodos de quarentena.

Para atingir os objectivos acima, no FastRank, optámos por usar puzzles criptográficos. Mais precisamente, quando um nó i contacta um nó j para estabelecer uma nova ligação, o nó j prepara um puzzle criptográfico que tem de ser resolvido por i para que j aceite a ligação. O tempo médio para resolver cada puzzle criptográfico corresponde ao tempo de quarentena pré-definido, considerando que todos os recursos computacionais de um nó estejam dedicados à sua resolução. Existem na literatura vários exemplos de puzzles criptográficos com estas propriedades [2, 17, 20]; no FastRank optámos por usar o mecanismo descrito em [5]. Uma vantagem desta abordagem é que um nó que se queira juntar à rede é limitado pelos seus próprios recursos, independentemente do número de nós diferentes a que ele se tenta ligar ou do número de identidades diferentes que tenta usar, porque o número de desafios que pode resolver por unidade de tempo é limitado pela capacidade do hardware. Desta forma, o FastRank mitiga também o impacto de outros comportamentos, como o ataque de Sybil [4, 19].

Os mecanismos acima descritos são assimétricos: enquanto o nó que tenta estabelecer a ligação tem de resolver um puzzle criptográfico, o nó que a aceita não

tem. A razão para este comportamento é que o FastRank é feito para detectar, e expulsar da rede, nós parasitas. Estes nós, que são expulsos, irão tentar estabelecer novas ligações, contactando diferentes membros. Para minimizar o impacto negativo que este requisito tem nos nós altruístas, se um nó recebe múltiplos pedidos de ligação concorrentemente, ele irá submeter um puzzle criptográfico diferente para cada nó que tente estabelecer a ligação e irá apenas ligar-se ao primeiro nó que termine a tarefa. Desta forma, se um nó parasita tentar resolver múltiplos puzzles criptográficos ao mesmo tempo, e competir com um nó altruísta pelas ligações, corre o risco de não ser aceite, dado que nós que dediquem todos os recursos a resolver um único puzzle têm maior probabilidade de responder primeiro e obter a ligação. Como um nó é obrigado a participar em repetidas interacções com os seus vizinhos imediatamente após a ligação ser estabelecida, caso se limite a consumir pacotes dessa ligação, irá ser detectado e irá perder a ligação antes de adquirir outra nova.

Mudámos também o comportamento do protocolo HyParView quando uma ligação é perdida. No protocolo original, um nó tenta pró-activamente manter a sua vista completa. Portanto, se um vizinho falha, um nó da sua vista passiva é escolhido e imediatamente usado para tentar substituir essa ligação. No entanto, no HyParView com Restrições, substituir uma ligação tem um custo. Além de que, se um nó tem poucas vagas na sua vista activa, é provável que múltiplos nós concorrentemente tentem ocupar essas vagas. Isto agrava o custo de um nó se juntar a uma rede onde todos os nós tentem encher a sua vista o mais cedo possível. Por outro lado, um nó altruísta pode optar por esperar novos pedidos de ligação por nós que se queiram juntar à rede para completar a sua vista activa. Neste caso, o nó altruísta não só evitará o custo de inicializar uma ligação, mas, também, facilitará o procedimento de ligação a novos nós altruístas que se queiram integrar. No HyParView com Restrições usamos um limite inferior, chamado *linha-de-água*, que tem de ser atingido antes de um nó pró-activamente procurar novos vizinhos. Se um vizinho falha ou é expulso (após ser detectado como parasita) mas o tamanho da vista activa é superior à *linha-de-água*, o nó simplesmente espera por pedidos, e irá aceitar ligações até que o tamanho máximo da *vista activa* seja alcançado.

A fonte da stream é tratada de forma diferente de todos os outros nós, na medida em que não mantém nenhuma vista activa explícita. Em vez disso, usa uma grande vista passiva para escolher de forma aleatória um número de pontos de contacto, cada vez que uma trama é enviada. Os nós altruístas que recebem a informação da fonte irão reenviá-la para os seus vizinhos (uma descrição pormenorizada é fornecida abaixo). O objectivo é distribuir a carga uniformemente pelos membros da rede, de modo a que não haja um grupo fixo de nós que estejam mais próximos da fonte (e que tenham sempre de retransmitir a informação) e outro grupo que se comporte sempre como folhas (e apenas receba informação).

3.2 Algoritmo de Classificação

O FastRank utiliza as propriedades da topologia gerada pelo HyParView com Restrições para executar um algoritmo localizado e eficiente, do tipo “olho-por-

olho", que consegue eficazmente detectar nós parasitas e, expulsá-los das vistas activas dos nós altruístas. O mecanismo é baseado na observação de que, normalmente, dois nós altruístas trocam a mesma quantidade de informação entre si. Se este balanço for bastante assimétrico, com grande probabilidade, o nó que está a receber e não está a enviar informação é um parasita.

O balanço das trocas com cada vizinho é capturado pelo FastRank através de uma classificação numérica atribuída ao vizinho [10]. Cada nó i mantém, para cada vizinho j da sua vista activa, uma classificação denominada $peso_{ij}$. O valor do peso de um novo vizinho é iniciado com um valor predefinido, chamado *peso base* e depois é mantido usando uma regra simples que consiste no incremento do peso do vizinho por uma unidade sempre que uma mensagem é recebida e no decremento do peso também por uma unidade quando uma mensagem é enviada para esse vizinho. No FastRank, o valor do *peso base* é 0. Isto significa que um vizinho que envia mais mensagens do que recebe tem uma pontuação positiva, e um parasita irá ter uma pontuação negativa. Além disso, o algoritmo de classificação também define um valor mínimo para a pontuação, chamado *peso mínimo*, abaixo do qual um nó é considerado parasita. Na Secção 5, discutimos que valor de *peso mínimo* deverá ser escolhido.

3.3 Disseminação das Mensagens

No FastRank, a disseminação das mensagens é concretizada da seguinte forma. A fonte escolhe, para cada pacote, um número f de contactos entre os nós presentes na rede (isto é conseguido mantendo uma vista passiva de grande dimensão) e depois envia a informação para esses nós. Quando um nó recebe um pacote, directamente da fonte ou dos seus vizinhos, primeiro confirma que a trama não é duplicada (para este propósito, cada nó guarda informação com os identificadores dos últimos pacotes que recebeu). Informação repetida é descartada e nunca re-enviada para nenhum vizinho. Se a trama é nova, o nó irá re-enviá-la para cada um dos seus vizinhos com uma probabilidade que é uma função do peso desse vizinho. O tamanho da vista activa é deliberadamente pequeno mas suficientemente grande para tolerar uma fracção de nós não altruístas. Como resultado, se todos os nós forem altruístas, recorrer a uma estratégia de inundação poderá originar bastantes mensagens redundantes. Deste forma, um nó altruísta irá enviar uma mensagem para outro nó altruísta com probabilidade inferior a 1 que se chama *probabilidade base de retransmissão* (ou simplesmente *pbr*). O valor de *pbr* é escolhido de forma a que a fiabilidade da disseminação seja alta mas com um custo reduzido que é imposto quando a inundação é usada. Naturalmente, o valor de *pbr* depende do tamanho da vista activa. Na Secção 5, discutimos como *pbr* pode ser configurado.

A probabilidade base de retransmissão é usada para disseminar mensagens para outros nós altruístas. Por outro lado, se um vizinho foi detectado como parasita (i.e., se o seu peso for inferior ao *peso mínimo*), então nenhuma mensagem é retransmitida para esse vizinho. Além disso, para vizinhos que têm um peso negativo mas superior ao limite *peso mínimo*, as mensagens são retransmitidas com uma probabilidade inferior a *pbr* mas superior a 0. Mais precisamente, o

FastRank usa a seguinte fórmula para calcular a probabilidade de retransmissão de um nó i para um vizinho j, denomidada $pr_i(j)$:

$$pr_i(j) = \begin{cases} pbr & \text{se } peso_{ij} \geq peso\ base \\ pbr \left| \frac{peso\ mínimo - peso_{ij}}{peso\ mínimo} \right| & \text{se } peso\ base > peso_{ij} \geq peso\ mínimo \\ 0 & \text{se } peso\ mínimo > peso_{ij} \end{cases}$$

4 Comportamentos Considerados

O FastRank foi concebido para sistemas onde a maioria dos nós é altruísta e apenas uma pequena fracção de nós são parasitas ou racionais. O caso em que todos os nós são racionais está fora do âmbito deste artigo. Como discutido no trabalho relacionado, este cenário requer protocolos consideravelmente mais onerosos. Na secção de avaliação, analizaremos a qualidade do sistema para diferentes fracções de nós da população que adoptam diferentes comportamentos, considerando que os nós pertencem a uma das seguintes categorias:

- *Nós altruístas:* Um nó altruista segue o protocolo especificado sem nunca adoptar um comportamento diferente.
- *Parasitas:* São nós que nunca reenviam qualquer mensagem. Recebem informação dos seus vizinhos até serem expulsos. Além disso, tentam maximizar o número de vizinhos que têm.
- *Nós Racionais:* São nós que se tentam desviar do protocolo para maximizarem a sua utilidade. Este comportamento é discutido abaixo.

4.1 Comportamento Racional

Consideramos que um nó racional obtém benefício sempre que recebe um novo pacote e tem um custo sempre que envia um pacote ou resolve um puzzle criptográfico. Portanto, para maximizar a sua utilidade, um racional tenta aumentar o rácio entre os pacotes recebidos e enviados. É claro da descrição do algoritmo que, se um nó não mantém as trocas de informação balanceadas com um vizinho, eventualmente será detectado como parasita e expulso por esse vizinho. Existem duas possíveis estratégias que um nó racional pode ter para aumentar a sua utilidade:

- *Diminuição da vista activa:* Um nó racional pode optar por diminuir a sua vista activa, de modo a que receba os conteúdos mas tenha de reenviar informação para um número inferior de vizinhos. Com este ataque o nó aparenta ser altruísta para os seus vizinhos, que não têm maneira de detectar que este está a usar uma vista reduzida. Algoritmos mais sofisticados (e.g. LiFTinG [7]) que não dependem somente de informação local, foram desenvolvidos para detectar este ataque.

– *Serviço mínimo (para manter a sua pontuação estritamente acima do limite):* Um nó racional pode reduzir a taxa à qual reenvia pacotes de maneira a que seja capaz de manter a sua pontuação acima do *peso mínimo* e, portanto, nunca ser expulso da vista activa dos nós altruístas. Além disso, pode tentar arranjar tantos vizinhos quanto possível. A ideia é que se tiver vizinhos suficientes, poderá receber a informação com boa qualidade, apesar de raramente reenviar a informação .

Vamos mostrar que, face aos desvios acima, o FastRank apresenta as seguintes propriedades: i) a qualidade recebida por nós altruístas é apenas parcialmente afectada, mesmo para uma grande fracção de nós com comportamento não altruísta; ii) os nós racionais ainda têm de contribuir com uma quantidade razoável de recursos para conseguirem receber os dados contínuos com qualidade aceitável; iii) se existir uma grande fracção de nós com comportamento racional, isto poderá ser detectado por nós altruístas.

5 Avaliação

Nesta secção, iremos reportar uma extensa avaliação do FastRank. Todas as experiências foram feitas usando o simulador PeerSim [18]. A fonte utiliza um protocolo baseado em ciclos, enquanto os restantes nós, utilizam um protocolo baseado em eventos. As simulações usaram 1000 nós e consistiram na disseminação de 20000 pacotes, sendo cada um injectado pela fonte em 7 nós escolhidos aleatoriamente. Os resultados apresentados nesta secção são a média de 100 testes independentes, usando as mesmas configurações. A avaliação está dividida em quatro partes. Na primeira, motivamos a escolha dos valores dos diferentes parâmetros do sistema. A segunda parte ilustra a operação quando todos os nós são altruístas. A terceira discute o efeito dos nós parasitas e, finalmente, a quarta o efeito dos nós racionais no sistema.

5.1 Configuração do FastRank

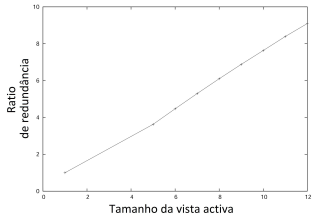
Os parâmetros que afectam a operação do FastRank são os seguintes: o tamanho da vista activa; a classificação mínima de um vizinho (*peso mínimo*); a probabilidade base de retransmissão *pbr*; e o *período de quarentena* (i.e., o tempo médio necessário para resolver um puzzle criptográfico quando se estabelece uma nova ligação). A Tabela 1 mostra os valores por omissão da configuração do FastRank. Discutimos a motivação para estes parametros nas subsecções seguintes.

Tempo de Criação da Rede. O tempo de criação da rede é o tempo necessário para que todos os nós acabem de resolver os puzzles criptográficos e estabeleçam um número de ligações igual ou acima da linha-de-água.

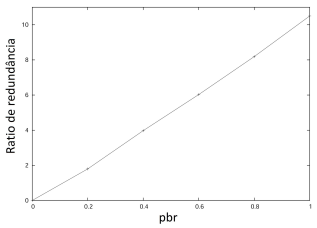
Tamanho da Vista Activa. Discutimos agora qual o tamanho da vista activa. Começamos por assumir que usariamos uma técnica de inundação para propagar as mensagens na rede (na secção seguinte discutimos porque é que não

parâmetro	valor	parâmetro	valor
vista activa	15	*pbr*	0.4
linha-de-água	12	período de quarentena (pacotes)	220
peso mínimo	-15		

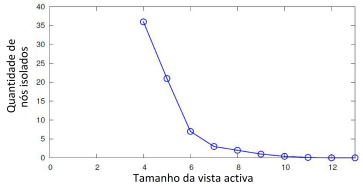
Tabela 1: Configuração por omissão doFastRank



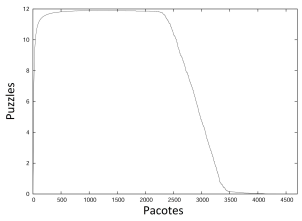
(a) Redundância vs tamanho vista.



(b) Redundância vs *pbr*.



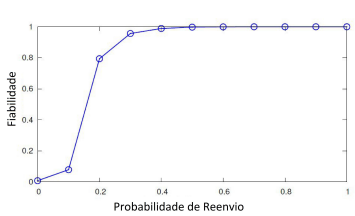
(c) Fracção de nós isolados após 30% de falhas.



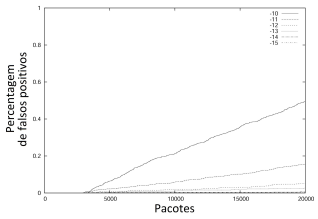
(d) Tempo para construir a rede.

Figura 1: Redundância, efeito de faltas e tempo para construir a rede

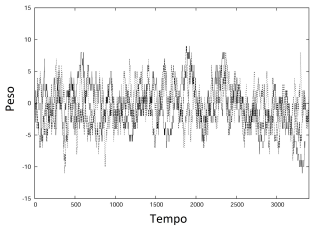
usamos inundação no FastRank). Se recorrêssemos à inundação, enquanto os nós altruístas estivessem ligados, todos estes nós iriam receber todas as mensagens. Desta forma, o tamanha da vista activa não teria impacto na fiabilidade. Portanto, o tamanho da vista activa deve ser escolhido de modo a que a probabilidade de que um nó altruísta fique isolado, na presença de nós parasitas, seja muito pequena. Optamos por configurar o FastRank para que pelo menos 30% de nós parasitas sejam tolerados com o mínimo dano para os nós altruístas. A Figura 1(c) mostra a percentagem de nós altruístas que ficam isolados após a construção da rede para diferentes tamanhos da vista activa. Como pode ser visto, se este tamanho for igual ou superior a 11, apenas 0.1% de nós altruístas ficam isolados. Este valor motivou o uso de uma abordagem mais conservativa e foi escolhido o valor 12.



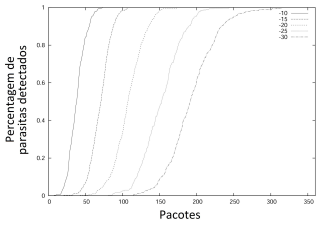
(a) Fiabilidade da difusão.



(b) Falsos-positivos em função do valor do peso mínimo.



(c) Flutuação do peso.

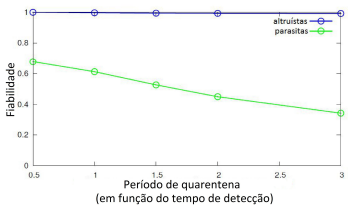


(d) Tempo necessário para detectar um parasita.

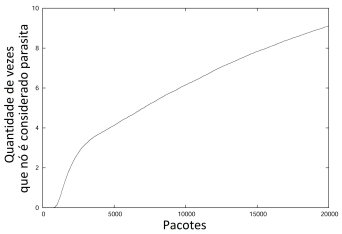
Figura 2: Parâmetros relevantes para a classificação dos vizinhos

Probabilidade de Reenvio. Na subsecção anterior foi escolhido um tamanho de vista activa em função da fracção de parasitas que pretendemos tolerar. Agora explicamos a motivação para escolher a probabilidade base de retransmissão *pbr*. A Figura 1(a) retrata quantas vezes, em média, um nó recebe a mesma mensagem se for usada inundação no sistema, para diferentes tamanhos da vista activa. Como pode ser visto, se a inundação for usada, existe bastante redundância, o que acarreta um desperdício significativo de recursos. Tal como nos protocolos epidémicos, evitamos esta redundância reenviando as mensagens com uma probabilidade inferior a 1. Fixando o valor da vista activa em 12, a Figura 2(a) retrata a fiabilidade do protocolo numa rede como a do FastRank, em função da probabilidade base de retransmissão e a Figura 1(b) retrata a redundância resultante. Como pode ser visto, escolhendo uma probabilidade base de retransmissão de 0.4 numa rede com uma vista activa de 12, é possível atingir valores de fiabilidade altos com uma redução significativa de redundância no processo de disseminação.

Classificação dos Vizinhos. O objectivo do mecanismo de classificação é detectar os nós parasitas o mais cedo possível, sem causar um número significativo de falsos positivos. Devido à aleatoriedade dos procedimentos de disseminação, o peso do vizinho poderá variar ao longo do tempo. Portanto, o valor de *peso mínimo* deverá considerar uma margem de segurança, para que estas variações



(a) Efeito do período de quarentena.



(b) Responde a um.

Figura 3: Configuração da quarentena

não marquem nós altruístas como parasitas. A Figura 2(c) mostra as variações do valor do peso entre dois nós altruístas para diferentes probabilidades base de retransmissão. Destes gráficos, é claro que o valor de *peso mínimo* não deverá ser superior a -15. De notar que, quanto mais baixo este valor for, menos provável é de gerar um falso positivo mas mais tempo demorará a detectar um nó parasita. Como discutimos na Secção 3, o protocolo penaliza nós com baixa pontuação, enviando-lhes mensagens com probabilidade inferior a *pbr*, prejudicando nós que enviam menos mensagens e conseguem manter a sua pontuação acima do *peso mínimo*. Um efeito secundário desta estratégia, é que o tempo necessário para detectar um nó parasita aumenta linearmente com o *peso mínimo*. A Figura 2(d) mostra o número de pacotes que um nó parasita recebe antes de ser detectado para diferentes valores de **peso mínimo**.

Período de quarentena. Como discutido antes, o objectivo do período de quarentena é assegurar que um nó parasita não consegue renovar as suas ligação mais rapidamente do que demora a ser detectado. A Figura 2(d) demonstra que são necessários 110 pacotes até detectar o último nó parasita, para um *peso mínimo* de -15. Desta forma, o mecanismo de entrada na rede deverá utilizar um puzzle criptográfico que, em média, demore mais tempo a ser resolvido do que demora a enviar esse número de pacotes. A Figura 3(a) mostra a fiabilidade de um nó com o seguinte comportamento: nunca reenvia nenhum pacote e tenta estabeleber quantas ligações quanto possível. Sabendo que enquanto estiver em período de quarentena não consegue estabelecer outra ligação, a figura retrata a fiabilidade obtida por um nó parasita para diferentes períodos de quarentena. Pode ser observado que se o período de quarentena for superior ao tempo que demora à sua detecção, então a proporção de pacotes recebidos por um nó parasita, decresce significamente.

Resolução do Puzzle Criptográfico. O período de quarentena assegura que um nó demora mais tempo a resolver um puzzle criptográfico do que aquele que um vizinho demora a detectar um nó como sendo parasita. Uma consequência desta abordagem é que se um nó apenas tem uma ligação e está a resolver puzzles para outros vizinhos, ele será considerado como parasita para o seu único vizinho,

antes de conseguir estabelecer novas ligações. A Figura 3(b) retrata quantas vezes, em media, um nó é marcado como parasita se este resolver um puzzle e logo após a sua resolução, envia a resposta. Como pode ser observado, cada nó, em média é considerado como parasita por 9 dos seus vizinhos.

5.2 Operação Livre de Desvios

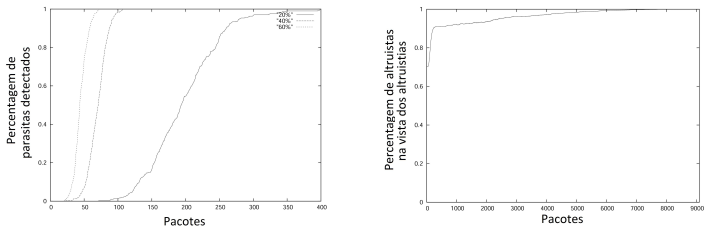
Nesta subsecção, fornecemos informação adicional sobre a operação num cenário livre de desvios, i.e., numa configuração onde todos os nós são altruístas. Para isto, considerámos uma transmissão contínua de dados de 24 tramas por segundo, geradas por uma única fonte, durante uma sessão completa de 14 minutos. A sessão começa com 1000 nós e a meio da transmissão (ao minuto 7), 100 novos nós são adicionados (i.e., neste momento, provocamos um aumento de 10% da população da rede). Com esta configuração, mostrámos: o tempo que demora a configurar a rede do FastRank, a fiabilidade recebida por um nó; o número médio de retransmissões de cada pacote; a percentagem de falsos-positivos durante a sessão; a quantidade de puzzles criptográficos resolvidos por cada nó; e, finalmente, o tempo que demora a um nó para se juntar a uma transmissão até atingir 90% de fiabilidade. Os resultados estão expostos na Tabela 2. Como pode ser visto, os valores obtidos coincidem com os previstos para a configuração do FastRank. Com uma vista activa de 12 e uma probabilidade base de retransmissão de 0.4, é esperado que cada trama seja envidada, em média, para 8.4 vizinhos, o que pode ser medido experimentalmente.

5.3 Tolerar Nós Parasitas

Nesta secção, fornecemos informação adicional sobre a operação do FastRank na presença de nós parasitas. Nestas experiências, o sistema corre com 100% de nós altruístas até ao momento em que uma fracção dos nós assume comportamento parasita. Estes nós deixam de reenviar informação e tentam estabelecer tantas ligações tantas possível. A Figura 4(b) mostra a evolução da composição das vistas activas dos nós após os parasitas terem sido injectados. A figura mostra que após 110 tramas os parasitas começam a ser detectados e o sistema começa a reconfigurar. Após, aproximadamente, 2500 tramas, o sistema atinge numa configuração onde 95% dos membros da vista activa de um altruísta são outros altruístas. Os parasitas ficam desligados da rede. Também medimos quantos puzzles criptográficos os altruístas e os parasitas têm de resolver durante a reconfiguração. Em média, um altruísta tem de resolver 6.75 puzzles antes de

Tamanho inicial da rede	1000	Falsos positivos	0.030
Nós adicionais	100	Fiabilidade global	0.999
Puzzles resolvidos pelos novos nós	12.2	Fiabilidade dos novos nós	0.995
Tempo para se juntar (pacotes)	3116		

Tabela 2: Operação com nós altruístas



(a) Tempo necessário para detectar um parasita para diferentes valores da *pbr* usando peso mínimo = -15.

(b) Recuperando de comportamentos parasitas.

Figura 4: Efeitos dos parasitas

estabilizar a sua vista activa com outros nós altruístas. Naturalmente, os parasitas continuam constantemente a resolver puzzles para substituir as ligações perdidas, mas isto não lhes permite receber a sequência de tramas.

5.4 Tolerar Nós Racionais

Esta subsecção, mostra os comportamento do FastRank na presença de nós racionais. Consideramos dois tipos de ataques descritos na Secção 4. Primeiro, discutimos o impacto de existirem nós que deliberadamente cancelam as ligações e aparentam comportarem-se correctamente para os nós que escolherem manter na sua vista. Experimentalmente, aferimos que os nós precisam de manter pelo menos 7 membros na sua vista activa para conseguerem ter uma fiabilidade aceitável. Desta forma, devido ao valor da *linha-de-água* usado pelos altruístas ser de 12 e o tamanho da *vista activa* ser de 15, os nós racionais conseguem poupar até 60% dos recursos. Na Tabela 3, mostramos a composição da vista dos nós altruístas e dos nós racionais, a fiabilidade da transmissão após o ataque, e o aumento na latência da transmissão devido ao ataque. Uma vez que os nós racionais limitam o número dos seus vizinhos, os nós altruístas substituem essas ligações por ligações a outros altruístas. Desta forma a rede mantém-se conexa e a fiabilidade não é afectada. No entanto, o diâmetro da rede aumenta, e isso tem um impacto observável na latência.

Finalmente, discutimos o impacto de existir um nó racional a enviar mensagens suficientes para manter a sua pontuação no limite de forma a nunca ser detectado. A Tabela 3 mostra o efeito deste tipo de comportamento. Como pode ser visto, se um nó racional utiliza este comportamento, mas não aumenta o número de ligações, é severamente penalizado. Por outro lado, mesmo com 30% de nós racionais, este ataque não tem impacto na fiabilidade observada pelos altruístas.

Também analisámos uma variante deste comportamento, onde não só os nós racionais mantêm as suas ligações no limite de *peso mínimo*, mas tentam esta-

Ataque de diminuição da vista activa: efeitos no sistema (30% de nós racionais)					
Vista activa dos nós racionais	-	*3*	*4*	*5*	*6*
Salto máximo	1	1.27	1.45	1.37	1.09
Salto médio	1	1.07	1.08	1.13	1.11
Fiabilidade altruístas	0.99	0.99	0.99	0.99	0.99
Fiabilidade racionais	-	0.03	0.06	0.67	0.88

Ataque de serviço mínimo: efeitos no sistema (30% de nós racionais)	
Fiabilidade dos altruístas após o ataque	0.97
Faibilidade dos racionais após o ataque	0.50
Fracção de mensagens enviadas pelos racionais	0.2

Ataque de serviço mínimo: efeitos no nó racional						
Vista activa dos nós racionais	*12*	*20*	*25*	*30*	*35*	*40*
Rácio de puzzles resolvidos	1	5	10	17	45	49
Rácio médio de mensagens	0.24	0.3	0.40	0.48	0.66	0.80
Fiabilidade do racional	0.71	0.80	0.86	0.98	0.94	0.99

Tabela 3: Efeitos de ataques racionais

belecer novas ligações. A Tabela 3 mostra quanto tempo um nó demora a ter uma fiabilidade semelhante à de um altruísta. Pode ser observado que um nó com um rank de *peso mínimo* em todas as suas ligações, precisa de manter uma vista activa constante de, pelo menos, 25. No entanto, tem de resolver 22 vezes mais puzzles criptográficos do que os nós altruístas para atingir esse ponto. Além disso, como tem de manter todas as ligações acima do limite, ainda é obrigado a reenviar informação: aproximadamente 80% de um altruísta. Desta forma, o ataque de serviços mínimos é menos lucrativo do que encolher o tamanho da vista activa. Estes resultados mostram que, apesar dos nós racionais beneficiarem de usarem estratégias sofisticadas num sistema como o FastRank, ainda têm de cooperar ajudando no processo de disseminação.

6 Conclusões

Neste artigo apresentámos o FastRank, um protocolo de transmissão contínua de dados entre-pares que se baseia na utilização de uma rede sobreposta com ligações simétricas para mitigar o efeito de nós parasitas de uma maneira eficiente e eficaz. O FastRank opera com baixo custo nos cenários favoráveis em que a percentagem de nós parasitas ou racionais é mínima (o que acontece regularmente na prática). O FastRank também consegue tolerar e detectar uma fracção de nós com comportamento racional, de forma a permitir reconfigurar o sistema quando o número de nós racionais se torna excessivo.

Agradecimentos Este trabalho foi parcialmente suportado pela Fundação para a Ciência e Tecnologia (FCT) através dos projectos com referência PEPITA (PTDC/EEI-SCR/2776/2012) e UID/CEC/50021/2013.

Referências

1. E. Adar and B. Huberman. Free riding on gnutella. *First Monday*, 5(10), October 2000.
2. Nikita Borisov. Computational puzzles as sybil defenses. In *IEEE P2P 2006*, pages 171–176, 2006.
3. Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, pages 68–72, 2003.
4. John R Douceur. The sybil attack. In *Peer-to-peer Systems*, pages 251–260. Springer, 2002.
5. Wu-chi Feng, E Kaiser, and Antoine Luu. Design and implementation of network puzzles. In *IEEE INFOCOM 2005*, volume 4, pages 2372–2382, 2005.
6. Ayalvadi J. Ganesh, A.-M. Kermarrec, and Laurent Massouli. Scamp: Peer-to-peer lightweight membership service for large-scale group communication. pages 44–55. Springer-Verlag, 2001.
7. Rachid Guerraoui, Kévin Huguenin, Anne-Marie Kermarrec, Maxime Monod, and Swagatika Prusty. Lifting: lightweight freerider-tracking in gossip. In *Middleware*, pages 313–333, 2010.
8. Yan Huang, Tom Fu, Dah-Ming Chiu, John Lui, and Cheng Huang. Challenges, design and analysis of a large-scale p2p-vod system. *SIGCOMM Comput. Commun. Rev.*, 38(4):375–388, 2008.
9. D. Hughes, G. Coulson, and J. Walkerdine. Free riding on gnutella revisited: The bell tolls? *IEEE Distributed Systems Online*, 6(6):1–, June 2005.
10. Murat Karakaya, Ibrahim Korpeoglu, and Ozgur Ulusoy. Free riding in peer-to-peer networks. *Internet Computing, IEEE*, 13(2):92–98, 2009.
11. A-M Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Trans. on Parallel and Distributed Systems,*, 14(3):248–258, 2003.
12. J. Leitão, J. Pereira, and L. Rodrigues. Epidemic broadcast trees. In *IEEE SRDS*, pages 301–310, Beijing, China, October 2007.
13. João Leitão, José Pereira, and Luís Rodrigues. Hyparview: A membership protocol for reliable gossip-based broadcast. In *IEEE DSN*, pages 419–428, 2007.
14. Harry C Li, Allen Clement, Mirco Marchetti, Manos Kapritsos, Luke Robison, Lorenzo Alvisi, and Mike Dahlin. Flightpath: Obedience vs. choice in cooperative services. In *OSDI*, volume 8, pages 355–368, 2008.
15. Harry C Li, Allen Clement, Edmund L Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin. Bar gossip. In *OSDI*, pages 191–204, 2006.
16. Xiaofei Liao, Hai Jin, Yunhao Liu, Lionel M Ni, and Dafu Deng. Anysee: Peer-to-peer live streaming. In *INFOCOM*, volume 25, pages 1–10, 2006.
17. Ralph C Merkle. Secure communications over insecure channels. *Comm. of the ACM*, 21(4):294–299, 1978.
18. Alberto Montresor and Mark Jelasity. Peersim: A scalable p2p simulator *.
19. Paul Resnick et al. The social cost of cheap pseudonyms. *Journal of Economics & Management Strategy*, 10(2):173–199, 2001.
20. XiaoFeng Wang and Michael K Reiter. Defending against denial-of-service attacks with puzzle auctions. In *Security and Privacy*, pages 78–92. IEEE, 2003.

Execução concorrente e determinista de transações

Tiago M. Vale[1], Ricardo J. Dias[1,2], João A. Silva[1], and João M. Lourenço[1]

[1] NOVA LINCS, Universidade Nova de Lisboa
[2] INESC-ID

Resumo Neste artigo apresentamos um protocolo de controlo de concorrência que garante que a execução concorrente de transações é equivalente à sua execução sequencial por uma ordem predefinida. Isto permite executar programas que usam transações de forma determinista. O protocolo (1) permite, pela primeira vez, a execução determinista de programas que usam memória transacional por *hardware*; e (2) garante a execução determinista de programas que usam memória transacional por *software* com um desempenho claramente superior ao estado da arte.

1 Introdução

Devido à massificação dos sistemas computacionais com múltiplos núcleos e/ou processadores, a programação concorrente com múltiplos fios de execução deixou de ser uma tarefa para especialistas e é agora uma tarefa essencial no processo de desenvolvimento de software. Nos programas sequenciais as operações são executada por um único fio de execução e pela ordem especificada no código fonte. Já no caso do programas concorrentes, a ordem pela qual operações de diferentes fios de execução executam é imprevisível e irá provavelmente variar entre execuções. Embora esta imprevisibilidade possa ser vista como um inevitável preço a pagar pelos benefícios da execução de múltiplos fios de execução, a imprevisibilidade da ordem de execução das operações dificulta a construção de programas confiáveis. Por exemplo, a imprevisibilidade da execução concorrente dificulta a reprodução de execuções e, portanto, o entendimento de comportamentos anómalos e o diagnóstico das sua causa; muitos destes erros apenas se manifestam quando certas operações de diferentes fios de execução executam numa determinada ordem [11].

A investigação em execução determinista de programas com múltiplos fios de execução tem sido focada em programas que usam trincos [1,4,10,12,14]. Embora os trincos sejam a técnica convencional utilizada para controlar acessos concorrentes ao estado partilhado por vários fios de execução, a Memória Transacional (MT) [8,18] é cada vez mais uma alternativa viável devido a um bom compromisso entre desempenho e complexidade de desenvolvimento. Com MT os programadores apenas precisam de especificar quais as sequências de operações que devem executar de forma indivisível (transações) sem se preocuparem em *como* garantir essa indivisibilidade. Esta é garantida em tempo de execução

através de um protocolo de controlo de concorrência implementado em *software* (MTS), *hardware* (MTH), ou numa mistura de ambos. A MT está a chegar ao programador comum: os processadores mais recentes da Intel e IBM já dispõem de MTH [3,23], o compilador GCC já possuí funcionalidades experimentais desde a versão 4.7 que permitem escrever programas com MT (utilizando MTS ou MTH) [6], e existem também esforços em curso para integrar construções de MT diretamente na linguagem C/C++ [2].

Os sistemas para execução determinista de programas que usam trincos podem, em princípio, ser usados para garantir a execução determinista de programas que usam MTS se, e só se, o protocolo de controlo de concorrência da MTS for implementado com trincos (deterministas). Contudo, esta abordagem tem alguns inconvenientes, nomeadamente: (a) não pode ser aplicada a MTH, porque o protocolo de controlo de concorrência está implementado no *hardware*; (b) perde a oportunidade de tirar partido da semântica transacional para reduzir os custos de garantir determinismo, porque este está a ser garantido pelo uso de trincos, que estão num nível de abstração inferior ao das transações; e (c) muitas MTS práticas usam primitivas atómicas de baixo nível, ex. *compare-and-swap*, em vez de trincos. Este artigo discute como garantir, eficientemente, a execução determinista de programas que usam MT. Nós advogamos a utilização de transações preordenadas como uma abordagem fundamental: estas fornecem a ilusão de que executam uma de cada vez, sempre numa ordem predefinida. Assim, propomos um novo protocolo de controlo de concorrência, aplicável tanto a MTS como MTH[3], que garante que o resultado da execução concorrente de um conjunto de transações é sempre equivalente a uma execução sequencial dessas transações numa ordem predeterminada. Este novo protocolo de controlo de concorrência, a que chamamos TPO, utiliza duas técnicas para garantir equivalência a uma ordem predefinida: *confirmação ordenada* e *modos de execução*. A confirmação ordenada força as transações a serem confirmadas de acordo com a ordem predefinida. Infelizmente a confirmação ordenada pode induzir desperdício de tempo porque uma transação pronta a ser confirmada tem de esperar que outra que a precede termine e seja confirmada primeiro. Para mitigar este problema tiramos partido da observação que, a todo o momento, existe sempre uma transação que é a próxima a ser confirmada. O protocolo TPO executa essa transação o mais rapidamente possível evitando tanto quanto possível os custos associados a controlo de concorrência (modo de execução *garantido*), enquanto que as restantes transações executam de acordo com um protocolo de controlo concorrência tradicional que garante a correção dessas transações na presença da transação rápida (modo de execução *especulativo*).

Implementámos dois protótipos do TPO: um que usa MTS e outro que usa MTH. A versão que usa MTH baseia-se num sistema de MTH disponível comercialmente e sem qualquer modificação. Avaliámos ambos os protótipos utilizando o popular banco de testes STAMP [13] e concluímos que o protocolo TPO avança o estado da arte tanto em *software* como em *hardware*. A versão *software* tem um desempenho claramente superior ao estado da arte (DeSTM [16]) e per-

[3] E sistemas transacionais em geral.

mite executar transações deterministicamente com baixo custo, demonstrando que usar simultaneamente transações e execução determinista é uma aproximação viável para facilitar o desenvolvimento de programas concorrentes. A versão *hardware* do protocolo TPO é, tanto quanto sabemos a primeira a permitir a execução determinista de transações utilizando um mecanismo de MTH disponível comercialmente, conseguindo-o com um custo modesto.

2 Visão geral

A opacidade [7] é o critério de correção utilizado em Memória Transacional (MT). Brevemente, a opacidade dá as mesmas garantias que *serializability* [15] com a restrição adicional que nenhuma transação pode observar estado inconsistente, nem mesmo as transações que venham a ser abortadas. As transações opacas são executadas concorrentemente mas aparentam executar uma de cada vez. Apesar disso a opacidade permite flexibilidade ao nível do protocolo de controlo de concorrência porque as transações podem aparentar executar por uma ordem *qualquer*. Os protocolos de controlo de concorrência tradicionais que implementam opacidade abraçam a flexibilidade dada por esta e realizam duas tarefas simultaneamente com a execução de transações: (a) computam a ordem na qual as transações aparentam executar (ordenação); e (b) controlam a execução concorrente das transações para respeitar essa ordem (controlo de concorrência). Como o processo de ordenação está entrelaçado com o controlo de concorrência, a ordem na qual as transações aparentam executar depende das intercalações imprevisíveis que ocorrem durante a execução concorrente de transações. Como tal, a ordem das transações pode, e costuma, mudar entre diferentes execuções do mesmo programa. Neste artigo apelidamos este modelo de execução de *transações tradicionais*.

Com *transações preordenadas* a ordem de execução é independente das intercalações que possam ocorrer em tempo de execução porque, ao contrário das transações tradicionais, as transações preordenadas, como o nome indica, já estão ordenadas *antes* de serem executadas. Conceptualmente as transações preordenadas passam por duas fases durante a sua execução: (1) a fase de *ordenação*, que define qual é a posição de todas as transações na ordem de execução; e (2) a fase de *execução*, onde as transações são executadas concorrentemente de tal maneira que o resultado final é equivalente à execução sequencial das transações na ordem predefinida. Os protocolos de controlo de concorrência tradicionais *não* podem ser usados na fase de execução porque implementam simultaneamente a ordenação e o controlo de concorrência. Neste artigo propomos um novo protocolo que pode ser usado na fase de execução.

2.1 Fase de ordenação

Uma consequência da separação da ordenação do controlo de concorrência é que a fase de ordenação e a fase de execução (onde o controlo de concorrência é feito) podem ser realizadas separadamente por dois componentes diferentes. A fase de

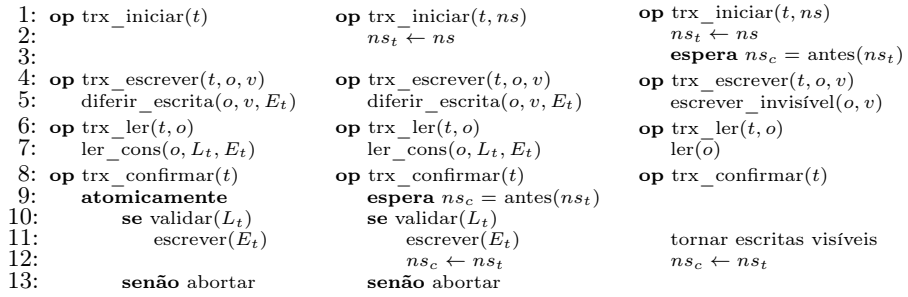
ordenação é efetuada por um *ordenador* que calcula uma ordem total sobre o conjunto de todas as transações. O ordenador é livre de computar qualquer ordem deterministicamente de acordo com critérios dependentes da aplicação. Na prática, para garantir a execução determinista de uma aplicação, o ordenador pode ordenar as transações à medida que estas estão a ser geradas pela aplicação, de uma forma determinista, para garantir que todas as execuções da aplicação resultam na mesma ordenação das transações. Por exemplo, o ordenador pode ordenar por turnos as transações geradas pelos vários fios de execução. Dado um programa com dois fios de execução t_1 e t_2, tal que o fio de execução t_1 executa as transações a_1 e b_1, e o fio de execução t_2 executa as transações c_2 e d_2, a ordenação seria $\langle a_1, c_2, b_1, d_2 \rangle$. Outra hipótese possível para o ordenador é usá-lo para reproduzir uma execução (não-determinista) anterior. Neste caso é necessário gravar a ordem pela qual os fios de execução confirmam cada transação na execução que se pretende reproduzir. Para reproduzir a execução o ordenador ordena as transações de acordo com a informação gravada.

2.2 Fase de execução

Uma vez ordenadas, as transações podem ser executadas. No coração da fase de execução está um protocolo de controlo de concorrência que garante equivalência à ordem definida na fase de ordenação. A implementação trivial desse protocolo é simplesmente executar as transações sequencialmente pela ordem indicada. Contudo essa estratégia é claramente insuficiente porque não tira partido do paralelismo inerente às arquiteturas com múltiplos processadores. Na próxima secção apresentamos o cerne deste artigo, um protocolo para ser usado na fase de execução que permite a execução concorrente de transações preordenadas.

3 Protocolo TPO

Nesta Secção descrevemos o TPO, um novo protocolo de controlo de concorrência que, embora execute transações concorrentemente, garante equivalência a uma execução sequencial dessas transações numa ordem predefinda. O TPO está desenhado como uma metodologia que pode ser aplicada a protocolos de controlo de concorrência otimistas [9], que funcionam da seguinte maneira. Uma transação otimista consiste numa, ou mais, execuções especulativas. Cada execução especulativa está dividida em três fases: (1) leitura; (2) validação; e (3) escrita. Durante a fase de leitura a transação regista no seu conjunto de leituras todos os objetos do estado partilhado que lê. As operações de escrita não modificam o estado partilhado; em vez disso a transação regista o novo valor no seu conjunto de escritas, diferindo as modificações para quando for confirmada. (Objetos que sejam lidos e escritos estão contidos tanto no conjunto de leituras como no de escritas.) Após a fase de leitura a transação passa por uma fase de validação onde verifica se algum objeto do seu conjunto de leituras foi modificado por uma transação concorrente. Se sim, então a transação é abortada para garantir opacidade, e pode ser reexecutada; caso contrário a transação passa para a fase de



(a) Otimista. (b) TPO: especulativa. (c) TPO: garantida.

Figura 1: Metodologia para transformar o protocolo otimista no TPO. A Figura 1a modela uma transação otimista típica. As Figuras 1b e 1c modelam o modo de execução especulativo e garantido do TPO, respectivamente. ns_c representa o número de sequência da última transação que foi confirmada. ns_t, L_t e E_t representam o número de sequência, conjunto de leituras, e conjunto de escritas da transação t, respectivamente.

escrita. Na fase de escrita a transação modifica todos os objetos pendentes no seu conjunto de escritas de forma indivisível.

Escolhemos um protocolo otimista como base para o TPO porque é apropriado para transações dinâmicas, isto é, transações para as quais é muito difícil (ou até impossível) identificar os seus conjuntos de leitura e escrita sem as executar. Transações dinâmicas são bastante comuns em programas que usam MT em linguagens de propósito geral devido ao *aliasing* e à natureza não-estruturada da *heap*. De facto, os protocolos de controlo de concorrência da MT, tanto por *software* como por *hardware*, são normalmente otimistas.

No resto desta Secção apresentamos o TPO de forma incremental. Primeiro descrevemos o protocolo otimista base na Secção 3.1, seguido da nossa metodologia para transformar o protocolo base no TPO através da aplicação de duas técnicas-chave: confirmação ordenada, na Secção 3.2, e modos de execução, na Secção 3.3.

3.1 Protocolo base

Consideremos o protocolo na Figura 1a que modela o esquema otimista. A fase de leitura ocorre depois da operação $trx\_iniciar$ e antes da $trx\_confirmar$, e consiste em invocações da trx_ler e/ou $trx\_escrever$. Tanto a fase de validação como a de escrita ocorrem durante a operação $trx\_confirmar$.

Fase de leitura. Operações de escrita com o objetivo de modificar o valor do objeto o para v registam essa informação no conjunto de escritas E_t (Figura 1a, linha 5). Uma operação de leitura no objeto o regista esse acesso no conjunto de leituras L_t e retorna o valor da última escrita pendente para o em E_t; caso não exista então retorna um valor de o no estado partilhado que seja consistente com o resto do conjunto de leituras (linha 7). Se não for possível retornar um valor de o consistente então a transação aborta. Por exemplo, imaginemos dois

objetos x e y, ambos inicialmente com o valor 0. A transação t observa $x = 0$. Entretanto, outra transação é confirmada e modifica tanto o valor de x como de y para 1. Se a transação t tentar ler y deve observar o valor 0 ou abortar, mas nunca poderá observar 1 porque $x = 0$ e $y = 1$ não é um estado possível de acordo com a opacidade.

Fase de validação. Na fase de validação a transação verifica se todos os objetos que observou ainda estão coerentes, isto é, se entretanto nenhuma transação os alterou (linha 10).

Fase de escrita. Se a validação for bem sucedida então a transação executa todas as modificações que tem pendentes no seu conjunto de escritas (linha 11).

Correção. Este protocolo garante opacidade devido à restrição de que as transações abortam se as leituras não puderem devolver um valor consistente e à indivisibilidade da execução das fases de validação e escrita (linhas 9–13). Se a fase de validação for bem sucedida então nenhum dos objetos lidos foi modificado desde a fase de leitura. Isto significa que a fase de leitura acontece logicamente no mesmo instante que a fase de validação. Como as fases de validação e de escrita executam de forma indivisível, a fase de escrita também acontece logicamente no mesmo instante que a fase de leitura. Como tal, a transação t aparenta ter executado sozinha no sistema, depois das transações que escreveram os valores que t observou e antes das transações que observarão os valores escritos por t.

3.2 Confirmação ordenada

O protocolo base (otimista) descrito na secção anterior dá a ilusão de que as transações executam uma de cada vez. Contudo, a ordem pela qual as transações aparentam executar é imprevisível porque depende da intercalação das operações das transações que ocorra durante a execução.

Para respeitar a ordem definida pela fase de ordenação, fazemos duas observações: (a) as transações otimistas só modificam o estado partilhado durante a sua fase de escrita; e (b) a posição de cada transação na ordem de execução depende da ordem relativa pela qual cada transação (atomicamente) executa a sua fase de validação e escrita. Portanto se restringirmos as transações a executarem as suas fases de validação e escrita na ordem definida pela fase de ordenação, garantimos que o resultado é equivalente à respectiva execução sequencial.

Para transformar o protocolo otimista descrito na secção anterior no TPO, começamos por alterar a operação $trx\_iniciar$ para ter um parâmetro adicional, um número de sequência ns, que reflete a posição dessa transação na ordem de execução definida pelo ordenador na fase de ordenação (Figura 1b, linha 1). A transação t está preordenada depois da transação com o número de sequência $antes(ns_t)$ e depois da transação com o número de sequência $depois(ns_t)$. Para forçarmos as transações a serem confirmadas de acordo com a ordem predefinida, inserimos uma espera condicional na operação $trx\_confirmar$. Quando a transação t quer ser confirmada, esta espera até a transação com o número de sequência $antes(ns_t)$ seja confirmada (linha 9). As transações comunicam entre elas através de um objeto ns_c cujo valor é o número de sequência da última transação a ser confirmada (linha 12).

Correção. No protocolo otimista original um dos principais fatores que garantem a sua correção é a execução indivisível da fase de validação e escrita das transações. Contudo a ordem pela qual as transações executam esse bloco indivisível depende da execução concorrente e imprevisível das transações. Para respeitar a ordem predefinida, no protocolo TPO esse bloco indivisível é substituído por uma espera condicional que restringe a ordem pela qual as transações são confirmadas. Concretamente, uma transação que acabe a sua fase de leitura só executa a sua fase de validação e escrita *após* a transação que a precede diretamente na ordem predefinida ter sido confirmada. Como as transações estão totalmente ordenadas, a condição de espera é verdade apenas para *uma transação de cada vez*. O protocolo continua a estar correto porque a espera condicional continua a garantir a indivisibilidade da fase de validação e escrita (o escopo de atomicidade é entre a espera condicional e a atualização de ns_c).

3.3 Modos de execução

O controlo de concorrência otimista usa uma série de técnicas para garantir a correção, nomeadamente conjuntos de leitura e escrita, validação, e escritas diferidas. Todas as transações são executadas utilizando essas técnicas porque *qualquer* transação *pode* vir a ser a próxima transação a ser confirmada. Contudo, a utilização dessas técnicas impõe computação extra quando comparado com uma execução sequencial sem qualquer controlo de concorrência. Com o TPO, ao contrário do protocolo base, a ordem de execução está predefinida. Como o TPO restringe a ordem pela qual as transação são confirmadas, estas podem agora ter que esperar pela sua vez, o que leva a uma perda de paralelismo em comparação com o protocolo base. Para mitigar esta perda de paralelismo, tiramos partido da observação que a qualquer momento existe sempre uma única transação, à qual chamados *garantida*, que é a próxima transação a ser confirmada. A transação garantida pode executar sem utilizar a maior parte das técnicas do protocolo base, evitando assim computação extra (e desnecessária) e acelerando a sua execução. Como tal, o TPO distinge entre dois tipos de transações: garantidas e especulativas.

Transação garantida. A transação garantida é a *única* transação ativa cujos antecessores já foram todos confirmados. Esta é a próxima, e única, transação que pode ser confirmada imediatamente. Esta transação pode ser executada de forma mais eficiente se fundirmos a fase de leitura com a fase de escrita e removermos completamente a fase de validação, eliminando assim a maior parte das técnicas do controlo de concorrência otimista e os seus custos associados. As transações garantidas executam de acordo com o protocolo da Figura 1c.

Fase de leitura. As operações de escrita já não são diferidas; em vez disso elas modificam o estado partilhado diretamente (linha 5). (Notamos que apesar disso a escrita tem que se manter invisível para *outras* transações até à confirmação da transação garantida, de forma a garantir que quaisquer outras transações vejam *todas* as escritas da transação garantida, ou nenhuma.) Como as modificações são feitas imediatamente durante a (agora combinada) fase de leitura e escrita, as operações de leitura são reduzidas a simplesmente ler o valor atual do

objeto sem efetuar quaisquer testes de consistência nem registo no conjunto de leituras (linha 7).
Fase de validação. As transações garantidas executam até ao fim sem qualquer interferência de outras transações, portanto a fase de validação é desnecessária.
Fase de escrita. A fase de escrita é feita implicitamente durante a fase de leitura devido à estratégia de escrever diretamente no estado partilhado, portanto esta fase também é eliminada. É apenas necessário tornar todas as escritas efetuadas visíveis atomicamente para as outras transações (linha 11).
Correção. O nosso argumento para a correção é o mesmo que para a técnica da confirmação ordenada. Contudo, uma transação garantida não executa a sua fase de leitura de forma especulativa, esperando pelo seu turno para transitar para fase de validação e escrita. Em vez disso, a transação executa a (agora combinada) fase de leitura e escrita quando já for a sua vez de ser confirmada. Visto que a transação tem, efetivamente, acesso exclusivo de escrita no estado partilhado até ao fim da sua execução, fundir as fases de leitura e escrita (substituindo as escritas diferidas por escritas diretas) e remover a fase de validação não afeta a correção do protocolo.
Transação especulativa. Uma transação cuja vez para ser confirmada ainda não chegou é uma transação especulativa. Estas executam de acordo com o protocolo com confirmação ordenada descrito na Secção 3.2, onde já discutimos a correção. Estas mantém-se corretas na presença da transação garantida porque as escritas desta última são feitos atomicamente.
Promoção durante a execução. Como as transações garantidas não pagam a maior parte dos custos associados ao controlo de concorrência, uma transação especulativa t que esteja a executar a sua fase de leitura pode mudar imediatamente para o modo garantido assim que $ns_c = ns_t$ seja verdade, i.e., seja a sua vez de ser confirmada. Para ser promovida de especulativa a garantida durante a execução, a transação t valida imediatamente a parte da fase de leitura que já executou. Se a validação for bem sucedida então a transação t aplica no estado partilhado todas as suas escritas que estejam diferidas, mas *não* atualiza o objeto ns_c. A partir deste ponto a transação t pode executar o resto da sua lógica como uma transação garantida. Caso contrário, ela aborta e pode reexecutar em modo garantido desde o início.
Múltiplas transações garantidas simultaneamente. Se soubermos mais informação sobre as transações podemos executar múltiplas transações garantidas em paralelo. Transações consecutivas que não leiam, ou escrevam, objetos escritos uma pela outra podem todas executar simultaneamente em modo garantido, porque o resultado final é independente da ordem pela qual elas são confirmadas.

4 Avaliação experimental

Todas as experiências foram executadas num IBM POWER 8 com 20 núcleos divididos por 2 *sockets* (10 núcleos em cada um) e um total de 128 GB de memória primária. As experiências executam com *simultaneous multithreading* desativado. Realçamos que a máquina tem uma arquitetura com acesso não uniforme

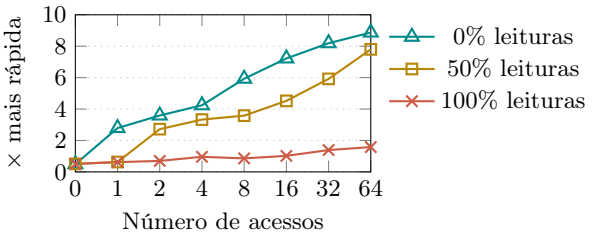


Figura 2: Quão mais rápida é uma transação TPO garantida que uma transação tradicional (em *software*).

à memória (NUMA). Concretamente, a latência de acesso à memória nas experiências é a seguinte: com 1–4 fios de execução a latência é uniforme; com 8 fios a latência aumenta até 2×, e com 16 fios aumenta até 4×.[4]

4.1 Memória transacional por *software*

Nesta secção apresentamos a avaliação do nosso protótipo que garante a execução determinista de transações em memória por *software*. Este protótipo foi implementado sobre o protocolo TL2 [5], ao qual aplicámos as transformações propostas na Figura 1. Pretendemos com esta avaliação responder às seguintes questões: (1) Quão eficazes são as transações garantidas? E (2) qual é o custo adicional no desempenho introduzido pelo protocolo TPO para garantir execução determinista? E como é que este se compara com o estado da arte?

Eficácia das transações garantidas. O objetivo das transações garantidas é reduzir o custo do controlo de concorrência na sua execução para mitigar a potencial perda de paralelismo induzida pela confirmação ordenada. Para medir quão eficaz é o modo garantido, executámos um banco de dados que consiste numa estrutura de dados chave-valor implementada com um vector de contadores. Executámos o banco de dados com um único fio de execução e fizemos variar (1) o número de acessos efetuado pelas transações; e (2) o rácio de leituras e escritas. A Figura 2 mostra quão mais rápida é a execução de uma transação TPO garantida comparativamente com a execução da mesma transação usando o protocolo de controlo de concorrência base. Uma transação com 0 acessos consiste na invocação de $trx\_iniciar$ imediatamente seguida de $trx\_confirmar$. Isto permite-nos medir o custo adicional imposto no desempenho pela nossa implementação da confirmação ordenada e modos de execução, que é de cerca de 2×. Aumentando o número de acessos feito pelas transação observamos que, como esperado, o modo garantido executa cada vez mais rápido que o protocolo base. Também observamos que as operações de escrita contribuem mais para o ganho do modo garantido: isto deve-se ao facto de que no protocolo base as operações de escrita impõem custos nas operações de leitura porque estas precisam de pesquisar no conjunto de escritas. As operações de escrita também contribuem

[4] Informação recolhida com o comando `numactl`.

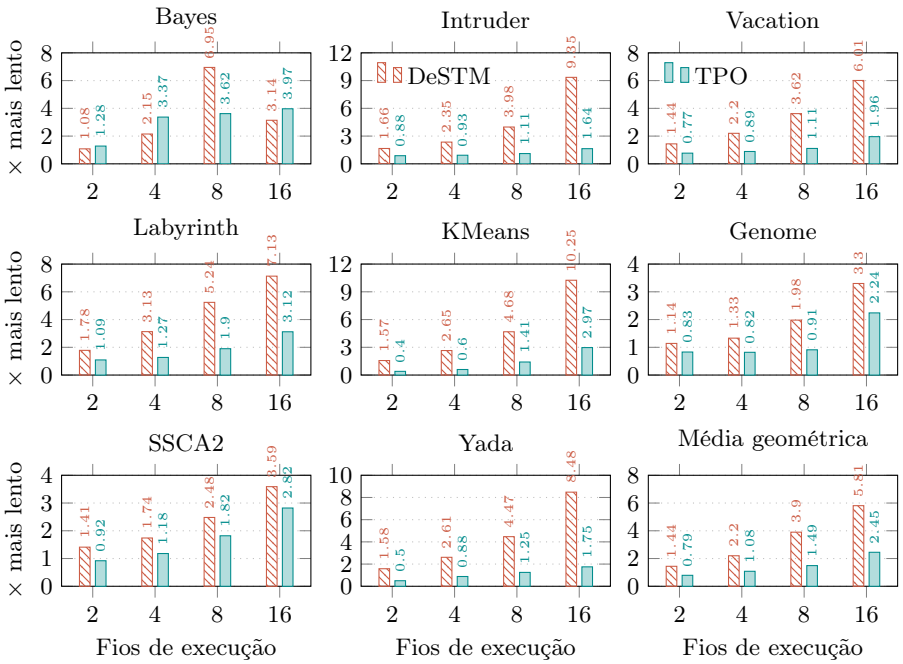


Figura 3: Custo de garantir execução determinista (em *software*) com o DeSTM e o TPO nos bancos de dados do STAMP. O eixo das ordenadas (yy) mede o tempo de execução de cada banco de dados normalizado para a execução não-determinista utilizando o protocolo base (menor é melhor). O canto inferir direito mostra a média geométrica, ou seja, a tendência geral no STAMP.

mais para o custo da confirmação da transação devido à necessidade de adquirir trincos para o conjunto de escritas antes de realizar as modificações pendentes e de posteriormente libertar todos os trincos adquiridos. As transações em modo garantido evitam todas estas fontes de computação adicional. Contudo, notamos que não existem ganhos observáveis quando as transações são apenas de leitura. A razão é simples: no TL2 as transações apenas de leitura não precisam de validar o seu conjunto de leituras para serem confirmadas. No geral, consideramos que o modo garantido é eficaz a minimizar os custos associados ao controlo de concorrência, mesmo em transações que efetuam poucos acessos: uma transação que efetua apenas uma leitura e uma escrita executa cerca de 3× mais rápido em modo garantido que em modo normal.

Comparação de desempenho com o estado da arte. Nesta secção avaliamos a execução determinista utilizando o TPO no popular banco de dados STAMP [13]. Este consiste em 8 aplicações representativas de diferentes domínios.Todos os resultados foram obtidos utilizando as configurações recomendadas e resultam da média de 5 execuções. Também comparamos contra o DeSTM [16], que representa o estado da arte de execução determinista com MTS.

DeSTM. Para realizar uma comparação honesta e significativa, implementámos o DeSTM no nosso próprio protótipo.[5] Tanto o DeSTM como o TPO são baseados no mesmo protocolo base e usam exatamente o mesmo ordenador. Uma das principais diferenças entre o DeSTM e o TPO é que neste último o ordenador estabelece uma ordem de serialização determinista das transações que o TPO garante, isto é, o resultado final é equivalente à execução sequencial das transações pela ordem definida pelo ordenador. Já o DeSTM usa o ordenador para definir uma ordem determinista entre os fios de execução na qual estes passam um testemunho que lhes permite tentar confirmar transações. Como tal, o resultado final é sempre equivalente a uma mesma ordem de serialização das transações, embora essa ordem seja desconhecida inicialmente. Como consequência deste desenho, o DeSTM ordena tanto as operações de aborto como de confirmação e requer que os conflitos entre transações sejam deterministas. Nós argumentamos que a necessidade de conflitos/abortos deterministas é um grande inconveniente, visto que a maior parte das implementações de MTS e todos os processadores com suporte para MTH padecem de falsos conflitos.

Desempenho. A Figura 3 quantifica o custo de garantir execução determinista utilizando o DeSTM e o TPO. O eixo das ordenadas (yy) mostra o tempo de execução dos bancos de dados normalizado para a execução não-determinista utilizando o protocolo base, dependendo do número de fios de execução (eixo das abcissas / xx). Nestes gráficos quanto menor o valor, melhor, e valores abaixo de 1 significam que a execução determinista foi *mais rápida* que a execução não-determinista tradicional. Realçamos quatro observações: (a) o custo de garantir determinismo aumenta com o número de fios de execução; (b) o TPO é melhor que o DeSTM em todos os bancos de dados independentemente do número de fios de execução, com a exceção do Bayes; (c) o TPO é no máximo $\approx 4\times$ mais lento que o protocolo (não-determinista) base enquanto que o DeSTM é até $\approx 10\times$ mais lento; e (d) o TPO chega a ser mais rápido que a execução não-determinista para algumas combinações banco de dados/fios de execução.

Não é surpreendente que o custo de garantir determinismo aumente com o número de fios de execução; a probabilidade de uma transação t querer ser confirmada antes do seu turno aumenta com o número de fios de execução, particularmente se existirem transações ordenadas antes de t que demorem mais tempo a executar que t. Ao contrário do DeSTM, o TPO minimiza estas situações utilizando transações garantidas para aumentar a probabilidade das transações não terem que esperar pela sua vez quando querem ser confirmadas. A Tabela 1 sustenta esta afirmação. Esta mostra, para cada combinação banco de dados/fios de execução, a percentagem do tempo de execução que as transações em média "desperdiçam" à espera que chegue a sua vez de confirmar. Observamos que em geral as transações TPO ficam menos tempo à espera da sua vez que as transações DeSTM, i.e., é mais frequente, com o TPO do que com o DeSTM, já ser

[5] O artigo do DeSTM aponta para um *URL* onde supostamente seriam disponibilizados os ficheiros executáveis da sua implementação, mas tal não se verificou até à data da submissão deste artigo. O nosso contacto com os autores do DeSTM no sentido de nos facultarem o acesso ao seu código não obteve qualquer resposta.

Tabela 1: Percentagem do seu tempo de execução que as transações passam à espera da sua vez para garantir determinismo, em média, utilizando o DeSTM e o TPO nos bancos de dados do STAMP.

Banco de dados	2 fios exec.		4 fios exec.		8 fios exec.		16 fios exec.	
	DeSTM	TPO	DeSTM	TPO	DeSTM	TPO	DeSTM	TPO
Bayes	42%	35%	66%	50–86%	80%	70–83%	92%	64–94%
Genome	19%	3%	37%	11%	66%	38%	85%	82%
Intruder	45%	21%	65%	45%	85%	73%	95%	89%
KMeans	40%	24%	67%	28%	84%	71%	94%	93%
Labyrinth	43%	8–45%	75%	50%	99%	48–69%	99%	49–84%
SSCA2	50%	10%	70%	61%	85%	88%	93%	96%
Vacation	23%	5%	46%	12%	68%	24%	82%	68%
Yada	45%	33%	70%	62%	84%	79%	96%	84%

a vez das transações serem confirmadas quando estas invocam $trx\_confirmar$. (O tempo gasto aumenta com o número de fios de execução, como esperado.)

Os resultados do Bayes são diferentes porque este é relativamente diferente dos restantes bancos de dados. No Bayes a ordem pela qual as transações são confirmadas no início da execução afeta dramaticamente o tempo de execução [17]. Portanto o comportamento é imprevisível, particularmente com o DeSTM dado que a ordem de serialização das transações é desconhecida. Tanto o DeSTM como o TPO usam sempre o mesmo ordenador[6]; não tentámos otimizar a ordenação para cada banco de dados.

Em conclusão, os excelentes resultados do TPO em comparação com o DeSTM devem-se à aceleração permitida pelo modo garantido, como vemos na Figura 2, e pela redução do tempo "perdido" à espera, como vemos na Tabela 1. O TPO marca assim um avanço significativo no estado da arte em termos de desempenho e apresenta evidências promissoras de que usar simultaneamente transações e determinismo para facilitar a programação concorrente é viável e poderá ser prático.

4.2 Memória transacional por *hardware*

Também avaliamos o nosso protótipo de MTH utilizando o STAMP. O suporte para MTH existente nos dias de hoje tem a particularidade de apenas garantir "o melhor que for possível" (*best effort*). Os processadores com MTH mantêm os conjuntos de leitura e escrita das transações apenas na *cache*; consequentemente, o número de operações que uma transação em *hardware* consegue efetuar está limitado pelo tamanho da *cache*. Transações por hardware que excedam os limites físicos do *hardware* nunca conseguirão executar com sucesso. É necessário fornecer *sempre* um fluxo de execução alternativo em *software* que garante a execução dessas transações. No nosso caso implementámos a solução mais comum: quando uma transação não consegue completar em *hardware* é executada em *software* com recurso a um trinco global que garante a execução em exclusão

[6] O ordenador ordena as transações dos fios de execução por turnos, tal como no artigo do DeSTM, p. ex., com 4 fios temos 1-2-3-4-1-2-3-4-1...

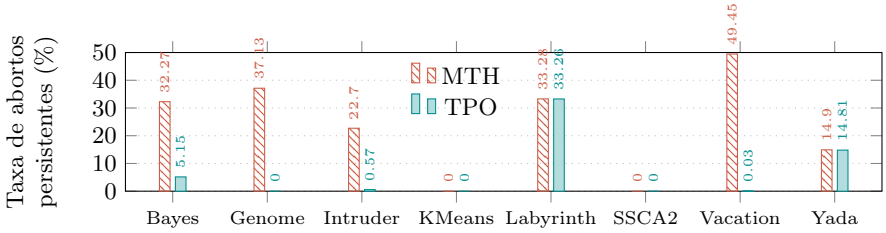


Figura 4: Taxa de abortos persistentes das transações tradicionais e transações TPO garantidas nos bancos de dados do STAMP (em *hardware*).

mútua com todas as outras transações, estejam essas a executar em *hardware* ou também em *software*.

Eficácia das transações garantidas. Enquanto que as transações garantidas em MTS permitem reduzir os custos associados ao controlo de concorrência, uma implementação em MTH que queira atingir o mesmo objetivo necessita de suporte do próprio *hardware*, coisa que não existe nos processadores atuais. Contudo, no nosso protótipo tiramos partido de um tipo específico de transações que a IBM fornece chamadas *rollback-only transactions* (*ROT*). As *ROT* são transações que apenas garantem que as suas escritas são indivisíveis, isto é, apenas mantêm o conjunto de escritas. Todas as leituras acedem diretamente à memória sem qualquer controlo de concorrência. Como as *ROT* só mantêm o conjunto de escritas, estas conseguem executar um maior número de acessos que uma transação normal. Contudo uma *ROT* pode ser abortada devido a conflitos de escrita-escrita na *cache*. No nosso protótipo as transações garantidas são executadas como *ROT*, o que lhes permite uma maior chance de executar completamente em *hardware*, não sendo necessário reverter para uma execução sequencial em *software*. Note-se que a transação garantida não pode executar em modo não-transacional porque isso não garantiria a atomicidade das suas escritas, o que violaria a opacidade.

Executámos cada banco de dados do STAMP com um único fio de execução, utilizando MTH tradicional e TPO e medimos o número de transações que excedem a capacidade do *hardware* e necessitam de reverter para o trinco global. A Figura 4 mostra que podemos dividir os bancos de dados em 3 classes. Na primeira classe temos o Labyrinth e o Yada, que possuem transações que não conseguem ser executadas em *hardware*. Na segunda classe temos exatamente a situação oposta: no KMeans e no SSCA2 todas as transações conseguem ser executadas em *hardware*. Finalmente temos a terceira classe com os restantes bancos de dados. Nesta classe existem transações que não conseguem executar em *hardware* utilizando o modo normal, mas conseguem executar como *ROT*s. Conseguimos ver claramente o benefício das transações garantidas: com o TPO a taxa de abortos é de cerca de 5% no Bayes e praticamente 0% nos restantes, enquanto que com MTH tradicional a taxa está acima de 20% em todos os bancos de dados. Em conclusão, as transações garantidas conseguem recuperar algum

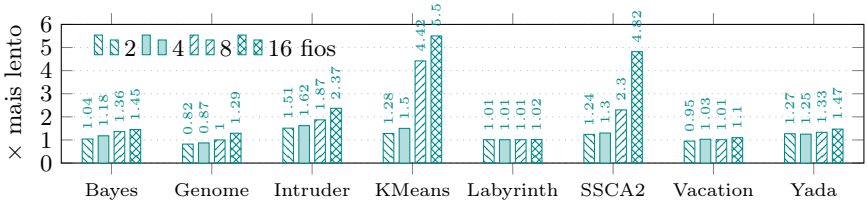


Figura 5: Custo de garantir execução determinista (em *hardware*) com o TPO nos bancos de dados do STAMP. O eixo das ordenadas (yy) mede o tempo de execução de cada banco de dados normalizado para a execução não-determinista utilizando o protocolo base (menor é melhor).

do paralelismo perdido devido à confirmação ordenada quando o protocolo base reverte para *software*.

Desempenho. A Figura 5 mostra o custo de garantir execução determinista utilizando o protótipo TPO em MTH. O desempenho do TPO é praticamente igual nos bancos de dados nos quais a execução tradicional reverte para *software* muitas vezes (Bayes, Genome, Labyrinth, e Vacation). Os resultados mais interessantes são os dos bancos de dados onde o protocolo base não necessita de reverter para *software* muitas vezes (Intruder, KMeans, SSCA2, e Yada). No Intruder e no Yada o TPO é apenas até $\approx 2\times$ mais lento que a execução não-determinista. O KMeans e o SSCA2, com transações pequenas e poucos conflitos, são ótimos para o protocolo base. Estas características tornam difícil mascarar os custos de garantir determinismo. Notamos que, como as transações garantidas não são aceleradas em MTH, e podem até abortar, existe uma descida significativa no desempenho de 4 para 8 fios de execução e ainda maior de 8 para 16, devido ao aumento da latência no acesso à memória da arquitetura NUMA.

Em conclusão, tanto quanto sabemos, o TPO avança o estado da arte porque permite pela primeira vez a execução determinista de programas que usam MTH. Em geral, o TPO garante execução determinista com um custo modesto, mas o aumento da latência no acesso à memória leva a uma perda de desempenho relativamente à execução não-determinista. No nosso protótipo as transações garantidas são implementadas como *ROT*s. Os resultados das transações garantidas no protótipo em *software* sugerem que suporte do próprio *hardware* para transações que *não* abortam permitirá diminuir os custos de garantir determinismo com MTH.

5 Trabalho relacionado

Transações preordenadas. A ideia de preordenar transações também é usada na comunidade de bases de dados com o objetivo de reduzir os custos das transações distribuídas [20]. Neste trabalho nós advogamos o seu uso para garantir execução determinista de programas baseados em MT. Os trabalhos referidos também descrevem um protocolo de controlo de concorrência para garantir que

uma ordem predefinida é respeitada. Contudo esse protocolo não pode ser usado no contexto do nosso trabalho, particularmente em MTH, porque utiliza uma estratégia pessimista e está desenhado para transações cujos conjuntos de leituras e escritas são determinados *a priori*. O TPO é otimista e mais geral porque funciona tanto com transações que tenham conjuntos de escritas e leituras estáticos como dinâmicos.

Execução determinista. Foram propostos variados sistemas para execução determinista de programas que usam trincos [1,10,14,4,12]. Se as transações forem implementados com trincos, executá-las deterministicamente pode ser feito utilizando esses sistemas como base. Contudo essa abordagem tem algumas limitações, nomeadamente: (a) não pode ser aplicada a MTH; (b) não tira partido da semântica transacional para reduzir os custos de garantir determinismo; e (c) muitas MTS práticas usam primitivas atómicas como *compare-and-swap* em vez de trincos. O DeSTM [16] adapta a técnica da dupla-barreira, usada por muitos sistemas que garantem a execução determinista de programas com trincos, para o contexto da MTS. O TPO estabelece uma ordem de serialização determinista e garante que a execução a respeita. O DeSTM define uma ordem determinista entre fios de execução pela qual estes trocam um testemunho que lhes permite confirmar transações. Como consequência, o DeSTM requer que os conflitos entre transações sejam deterministas (o que impossibilita que seja usado em MTH), enquanto que o TPO funciona independentemente disso.

Modos de execução. Executar uma transação com a garantia que ela não aborta é usado para permitir a utilização de *I/O* dentro de transações [19,22], e para melhorar o desempenho de MTS com poucos fios de execução [21]. As transações garantidas do TPO são semelhantes em espírito a estes trabalhos, contudo o seu objetivo é minimizar os custos de garantir determinismo. Ao contrário dos trabalhos referidos, múltiplas transações garantidas podem executar simultaneamente em paralelo tirando partido da existência de uma ordem predefinda, p. ex., uma sequência de transações que não tenham conflitos leitura-escrita e escrita-escrita entre si podem todas executar simultaneamente em modo garantido.

6 Conclusão

Neste artigo apresentámos o TPO, um protocolo de controlo de concorrência que faz uso do conceito de transações preordenadas para garantir a execução determinista de programas que usam múltiplos fios de execução e MT. A execução determinista é garantida utilizando duas técnicas: confirmação ordenada e modos de execução. Este artigo avança o estado da arte porque: (1) garante a execução determinista de programas que usam MTS com um desempenho claramente superior ao estado da arte, evidenciando que utilizar MTS e execução determinista para simplificar a programação concorrente é uma aproximação viável e poderá ser prático; e (2) permite, pela primeira vez, a execução determinista de programas que usam MTH, mas sugere que é desejável um melhor suporte do *hardware* para as transações garantidas.

Referências

1. T. Bergan, O. Anderson, J. Devietti, L. Ceze, and D. Grossman. CoreDet: A compiler and runtime system for deterministic multithreaded execution. In *ASPLOS*, 2010.
2. H. Boehm, J. Gottschlich, V. Luchangco, M. Michael, M. Moir, C. Nelson, T. Riegel, T. Shpeisman, and M. Wong. Transactional language constructs for C++. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3341.pdf, 2012.
3. H. Cain, M. Michael, B. Frey, C. May, D. Williams, and H. Le. Robust architectural support for transactional memory in the POWER architecture. In *ISCA*, 2013.
4. H. Cui, J. Simsa, Y. Lin, H. Li, B. Blum, X. Xu, J. Yang, G. Gibson, and R. Bryant. Parrot: A practical runtime for deterministic, stable, and reliable threads. In *SOSP*, 2013.
5. D. Dice, O. Shalev, and N. Shavit. Transactional locking II. In *DISC*, 2006.
6. GCC. Changes in GCC 4.7. https://gcc.gnu.org/gcc-4.7/changes.html, 2012.
7. R. Guerraoui and M. Kapalka. On the correctness of transactional memory. In *PPoPP*, 2008.
8. M. Herlihy and J. Moss. Transactional memory: Architectural support for lock-free data structures. In *ISCA*, 1993.
9. H. Kung and J. Robinson. On optimistic methods for concurrency control. *ACM TODS*, 6(2), 1981.
10. T. Liu, C. Curtsinger, and E. Berger. Dthreads: Efficient deterministic multithreading. In *SOSP*, 2011.
11. S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from mistakes: A comprehensive study on real world concurrency bug characteristics. In *ASPLOS*, 2008.
12. T. Merrifield and J. Eriksson. Conversion: Multi-version concurrency control for main memory segments. In *EuroSys*, 2013.
13. C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun. STAMP: Stanford transactional applications for multi-processing. In *IISWC*, 2008.
14. M. Olszewski, J. Ansel, and S. Amarasinghe. Kendo: Efficient deterministic multithreading in software. In *ASPLOS*, 2009.
15. C. Papadimitriou. The serializability of concurrent database updates. *JACM*, 26(4), 1979.
16. K. Ravichandran, A. Gavrilovska, and S. Pande. DeSTM: Harnessing determinism in STMs for application development. In *PACT*, 2014.
17. W. Ruan, Y. Liu, and M. Spear. STAMP need not be considered harmful. In *TRANSACT*, 2014.
18. N. Shavit and D. Touitou. Software transactional memory. *Dist. Comp.*, 10(2), 1997.
19. M. Spear, M. Silverman, L. Dalessandro, M. Michael, and M. Scott. Implementing and exploiting inevitability in software transactional memory. In *ICPP*, 2008.
20. A. Thomson, T. Diamond, S. Weng, K. Ren, P. Shao, and D. Abadi. Calvin: Fast distributed transactions for partitioned database systems. In *SIGMOD*, 2012.
21. J. Wamhoff, C. Fetzer, P. Felber, E. Rivière, and G. Muller. FastLane: Improving performance of software transactional memory for low thread counts. In *PPoPP*, 2013.
22. A. Welc, B. Saha, and A. Adl-Tabatabai. Irrevocable transactions and their applications. In *SPAA*, 2008.
23. R. Yoo, C. Hughes, K. Lai, and R. Rajwar. Performance evaluation of Intel® transactional synchronization extensions for high-performance computing. In *SC*, 2013.

Otimização do *HBase* para dados estruturados

Francisco Neves, José Pereira, Ricardo Vilaça e Rui Oliveira

INESC TEC & Universidade do Minho
francisco.t.neves@inesctec.pt, {jop,rmvilaca,rco}@di.uminho.pt

Resumo Os sistemas *NoSQL* escalam melhor que os tradicionais sistemas relacionais, motivando a migração de inúmeras aplicações para sistemas NoSQL mesmo quando não se tira partido da estrutura de dados dinâmica por eles fornecida. Porém, a consulta destes dados estruturados tem um custo adicional que deriva da flexibilidade dos sistemas *NoSQL*. Neste documento, analisa-se esse custo no *Apache HBase* e apresenta-se o *Prepared Scan*, uma operação adicional de consulta que visa tirar partido do conhecimento da estrutura de dados por parte da aplicação, diminuindo assim o custo associado à consulta de dados estruturados. Recorrendo à ferramenta de *benchmarking* YCSB, verifica-se que esta solução tem um aumento no débito de aproximadamente 29%.

Keywords: NoSQL, HBase, dados estruturados, desempenho

1 Introdução

Ao longo dos últimos anos, a quantidade de informação que tem sido requisitada pelos mais diversos clientes tem crescido de forma exponencial. Este facto levou a que os tradicionais sistemas de bases de dados relacionais (RDBMS) revelassem as suas limitações. Tal acontecimento encorajou a idealização e a conceção de novos sistemas.

As bases de dados *NoSQL* [1], também denominadas de não-relacionais, são então apresentadas como soluções capazes de acompanhar o crescimento dos dados e atacar os problemas que as bases de dados relacionais tornaram cada vez mais evidentes. Estes sistemas são executados numa configuração distribuída com múltiplos nós, pelos quais a informação total é dividida de acordo com uma estratégia bem definida em cada um dos sistemas. À técnica da divisão dos dados dá-se o nome de particionamento e é realizada de forma transparente para a aplicação, contrariamente ao que se verifica nos RDBMS devido à sua impraticabilidade e custos associados [2].

Para atingir uma solução capaz de se enquadrar no processamento analítico em larga escala, foi necessário descartar algumas das propriedades que caraterizam as bases de dados relacionais. Surgiu uma nova forma de aceder aos dados que se limita a um conjunto de operações simples: `get`, `put`, `scan` e `delete`, em detrimento da usual linguagem SQL. Operações adicionais podem ser fornecidas por alguns sistemas, como operações atómicas de incrementos. A desnormalização dos dados traduz-se na duplicação de partes dos mesmos [3] em múltiplas

tabelas, por forma a evitar a necessidade de agregações e operações `join`, e tornar as leituras mais rápidas. Esta é de facto uma característica dos sistemas *NoSQL* que implica o redesenho do esquema de dados de forma distinta da que era realizada nos RDBMS. Com o intuito de promover uma maior flexibilidade na forma como os dados são guardados, estas soluções não têm por base uma estrutura dos dados bem definida e permitem que parte desta, como as colunas, seja arbitrária e ajustável à medida que ocorre a manipulação de dados.

Apesar de nem todas precisarem desta flexibilidade, há aplicações que migram os seus dados dos sistemas relacionais para *NoSQL*, tendo como alvo exclusivamente a sua escalabilidade. Esta migração é facilitada por tecnologias que oferecem uma linguagem de interrogação semelhante a SQL sobre os sistemas *NoSQL*, como *Apache Hive* [4], *Apache Phoenix* [5] e *Cloudera Impala* [6] sobre o *Apache HBase*. Esta linguagem é convertida pelo próprio sistema em chamadas nativas ao sistema de gestão e processamento de dados *NoSQL* [7,8,9,10]. Estas ferramentas evitam assim a maior parte do esforço necessário para a migração para o contexto não-relacional [11].

Por outro lado, ao armazenarem dados que têm uma estrutura regular, nomeadamente, tendo todas as linhas as mesmas colunas, esta migração leva a ineficiências tanto no armazenamento como na consulta. Em particular, efetua-se repetidamente o armazenamento e transmissão da meta-informação relativa ao nome das colunas, pois o sistema não tem forma de antecipar que essa meta-informação é igual para todas as linhas. Isto é um problema importante no *Apache HBase*, apesar de ser o mais popular e integrar algumas as maiores aplicações, como Facebook [12].

Contribuição Neste documento, é apresentado o *Prepared Scan* para o *Apache HBase*, que visa otimizar os dados retornados, quando a aplicação conhece efetivamente a estrutura dos dados que lhe está associada, na operação `scan`, sendo esta a operação de leitura mais comum e que torna mais evidente este problema.

Estrutura do documento A Secção 2 deste documento apresenta uma contextualização dos sistemas *NoSQL*, em particular do *Apache HBase*. A Secção 3 foca-se na descrição da implementação e avaliação do desempenho do *Prepared Scan* face à operação `scan`. Na Secção 4, são apresentados os resultados obtidos e, por fim, na Secção 5 as conclusões.

2 Contextualização

NoSQL engloba uma extensa diversidade de tecnologias de bases de dados desenvolvidas para responder ao crescimento do volume de dados relativamente a utilizadores, produtos e objetos, à frequência a que esta informação é acedida e, também, às necessidades de desempenho e processamento desta. Ao longo deste documento o foco é direcionado para o *Apache HBase*, visto que é uma das tecnologias mais usadas e que têm tido mais sucesso.

2.1 *Apache HBase*

Apache HBase [13] é uma base de dados não-relacional, distribuída e escalável. Inspirada na *BigTable* da *Google* [14], é idealizada como um *sorted map* multi-dimensional indexado através do tuplo *(row key, column name, timestamp)*. Tanto o identificador de linha como o valor do tuplo são um conjunto de *bytes* não interpretados. O *sorted map* pode ter um número arbitrário e dinâmico de colunas que, por sua vez, se encontram agrupadas em famílias de colunas. A identificação de uma coluna é dada pela concatenação de uma família e de um qualificador de valor, isto é, *family:qualifier*, sendo a família de colunas um conjunto de caracteres legível e o qualificador um *byte array* opcional. Tipicamente as famílias de colunas são definidas de forma estática, contrariamente aos qualificadores que podem ser regularmente criados enquanto o sistema se encontra em execução.

O conjunto de pares chave-valor existentes é repartido horizontalmente resultando em regiões, a unidade de escalabilidade do *Apache HBase*. Estas regiões são, entretanto, distribuídas por diversos servidores denominados por *RegionServers*.

Cada região é persistida no sistema de ficheiros *Hadoop Distributed File System* (HDFS) [15], inspirado no *Google File System* (GFS) [16], e as instâncias que os gerem são denominadas *DataNodes*. Tipicamente, estas instâncias, com o intuito de maximizar a localidade, são executadas no mesmo nó que o *RegionServer*. No entanto, não invalida a possibilidade de ser atribuída uma região que não está no mesmo nó que o *RegionServer*.

Pares chave-valor O tipo fundamental do *Apache HBase* é o par chave-valor. O formato deste par é visível na Figura 1 e corresponde, com alguns campos de controlo, ao tuplo usado para indexar os valores. Quer as operações de escrita quer as de leitura lidam com pares chave-valor.

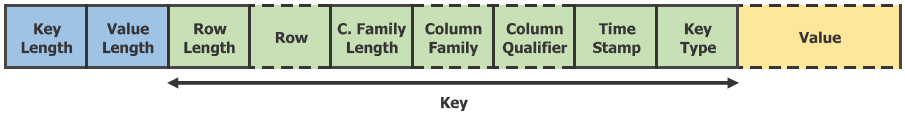


Figura 1: Formato do par chave-valor do *Apache HBase*

Como exemplo, considere que os pares chave-valor na Figura 2, outrora inseridos através da operação `put`, existem na base de dados. Estes pares são internamente guardados e mantidos com uma ordem lexicográfica dos identificadores de linha. A ordem total é dada primeiramente pelos identificadores de linha, seguidos do nome da família de colunas, seguido do qualificador e, por fim, pela marca temporal (em ordem inversa).

É possível obter uma representação destes dados recorrendo a uma tabela, se se assumir que cada par chave-valor é uma célula que resulta da interseção

```
(row-1, colfam1:qual1, 1294065304642) -> value
(row-1, colfam1:qual2, 1294065304642) -> value
(row-1, colfam2:qual1, 1294065304642) -> value
(row-2, colfam1:qual1, 1294065308222) -> value
(row-2, colfam2:qual1, 1294065303242) -> value
(row-3, colfam1:qual2, 1294065308222) -> value
```

Figura 2: Exemplo de pares chave-valor.

do identificador de linha com a coluna envolvente. A Tabela 1 é um exemplo dessa representação. As células em branco representam a inexistência de qualquer valor que resulta da interseção do identificador de linha com a coluna. O campo *timestamp* é uma marca temporal que pode ser representado como mais uma dimensão na tabela, uma vez que o sistema suporta múltiplas versões de células.

Tabela 1: Representação lógica de pares chave-valor do *Apache HBase*

	colfam1		**colfam2**
	qual1	**qual2**	**qual1**
row-1	value	value	value
row-2	value		value
row-3		value	

Interface com o cliente O acesso aos dados é facultado através de um conjunto de operações que são caraterísticas das estruturas de dados que lidam com pares chave-valor. Todas as alterações efetuadas através destas operações no *Apache HBase* são imediatamente visíveis em todas as leituras subsequentes, o que lhe confere um modelo de consistência forte.

`Get` Obtém pares chave-valor de uma determinada linha;
`Put` Insere um par chave-valor numa linha nova ou existente;
`Scan` Obtém múltiplas células de todas as linhas ou de um intervalo de identificadores de linha;
`Delete` Remove pares chave-valor de uma ou mais linhas.

Estas são as operações existentes geralmente em todos os sistemas *NoSQL*. No entanto, o *Apache HBase* em particular oferece operações *read-modify-write* que aplicam diversas instruções de forma atómica a uma linha:

`IncrementColumnValue` Incrementa um valor numa coluna[1];

[1] Um valor que não seja do tipo *big-endian long* pode resultar numa exceção.

`CheckAndPut` Verifica se o valor de uma linha, família ou de um qualificador corresponde ao esperado e, em caso afirmativo, executa a operação `put`.

Funcionamento do *Scan* Aquando da execução da operação `Scan` do *Apache HBase*, todos os seus parâmetros são enviados através de uma chamada RPC ao(s) *RegionServer(s)*. Até obter todos os resultados, várias chamadas podem ser realizadas. A quantidade de chamadas depende do tamanho da *cache* e das linhas a transferir, isto é, será maior no caso em que o valor do primeiro seja muito baixo e o número de linhas a transferir elevado. Os resultados retornados são pares chave-valor transmitidos com toda a informação presente na 1.

Coprocessors O *Apache HBase* permite a execução de código arbitrário no sistema, através do uso de *coprocessors*. São disponibilizados dois tipos de *coprocessors*: *observers* e *endpoints*. Os *observers* despoletam uma determinada ação quando um comportamento do sistema é verificado. Por sua vez, os *endpoints* permitem a extensão do protocolo e execução de código remoto, através de *Remote Procedure Call* (RPC), abstraindo o cliente da necessidade de ter em conta os característicos problemas dos sistemas distribuídos.

Alojados em cada *RegionServer*, são equivalentes aos *stored procedures* das conhecidas bases de dados relacionais e podem lidar diretamente com cada região. Desde a versão `0.94.2`, existe em cada *RegionServer* uma estrutura partilhada do tipo `Map<String, Object>` entre as várias instâncias de um determinado *coprocessor* que armazena estado que se pretende manter entre chamadas RPC.

3 *Prepared Scan*

A implementação da operação `scan` no *Apache HBase* tem como opção delimitar os resultados às colunas pretendidas, recorrendo ao método `addColumn( byte[] family, byte[] qualifier )`.

O *Prepared Scan* é uma operação de consulta de dados, baseada no `scan` nativo, que estende o protocolo de comunicação entre cliente e servidor do *Apache HBase* por meio da implementação de um *coprocessor* do tipo *endpoint*, ou seja, sem alteração do código-fonte do sistema. Esta operação é dividida em três fases:

Preparação Envio do conhecimento existente por parte da aplicação para os *RegionServers* e preparação do ambiente para a execução da operação;
Execução Obtenção de resultados previamente tratados de acordo com o conhecimento sobre os dados;
Conclusão Finalização da operação e libertação de todos os recursos usados durante as execuções desta.

Nesta secção será primeiramente enunciada a interface disponível para o cliente e como, através desta, o conhecimento é transmitido pela aplicação. De seguida, o protocolo usado nesta operação em comparação com o da operação `scan` nativa e, por fim, o funcionamento do *coprocessor*.

3.1 Interface com o cliente

A operação é representada pelas assinaturas dos métodos presentes na Figura 3, que refletem as três fases: preparação, execução e conclusão. Na fase de preparação, é dado como parâmetro um objeto do tipo *Scan*, uma instância da operação `scan` nativa que se pretende executar por forma a tirar proveito do conhecimento existe neste e, para tal, a instância desta deve conter as colunas que são pretendidas. É importante realçar que os parâmetros usados na fase de execução são usados da mesma forma que o `scan` nativo, isto é, o `startRow` e `stopRow` delimitam o início (inclusive) e fim (exclusive) do intervalo de identificadores de linha onde ocorre a consulta. O parâmetro `caching` indica a quantidade de linhas que serão mantidas em *cache* no cliente. É possível estender a implementação, numa fase posterior, de modo a suportar filtros entre execuções.

```
public interface PreparedScan extends Closeable {
        public PreparedScan prepare( Scan scan );
        public ResultScanner execute( byte[] startRow, byte[] stopRow,
                int caching );
        public void close();
}
```

Figura 3: Métodos das fases do *Prepared Scan*

Esta interface do cliente permite que sejam efetuadas múltiplas execuções com a mesma instância *Scan*, sendo alterados apenas estes três parâmetros entre cada execução, o que se traduz num reaproveitamento de todas as opções definidas inicialmente, como filtros, colunas a retornar, entre outros. A Figura 4 demonstra um exemplo de utilização do *Prepared Scan* com as colunas presentes na Tabela 1. Por motivos de espaço, considere que todos os conjuntos de caracteres são convertidos implicitamente para *byte array*. Note que são realizadas duas execuções com parâmetros diferentes e que, durante estas, não é necessário redefinir as colunas que se pretendem.

3.2 Protocolo

O protocolo usado no *Prepared Scan* é implementado recorrendo a *Protocol Buffers* (PB), uma vez que é a biblioteca de serialização de estruturas de dados definida pelo *Apache HBase*.

A fase de preparação, como foi referido, consiste no envio dos parâmetros, através de chamadas RPC realizadas em paralelo, da instância *Scan*, à exceção do `startRow`, `stopRow` e `caching`, a os *RegionServers* que, por sua vez, guardam esta informação associada ao cliente. É necessário o envio destes dados todos os *RegionServers* devido à inexistência de conhecimento prévio da localização dos resultados da operação. Note que esta chamada RPC inicial não é efetuada na

```
// Declare configuration, table's name and prepared scan
Configuration config = ...;
TableName table = ...;
PreparedScan pScan = ...;

Scan scan = new Scan();
scan.addColumn( "colfam1", "qual1" );
scan.addColumn( "colfam1", "qual2" );
scan.addColumn( "colfam2", "qual1" );

// Prepare
pScan.prepare( scan );

// Execute one or more times
ResultScanner scanner = pScan.execute( "row1", "row2" );
for( Result res : scanner ) { ... }

ResultScanner scanner = pScan.execute( "row3", null, 10 );
for( Result res : scanner ) { ... }

// Free resources
pScan.close();
```

Figura 4: Exemplo de execução de um *Prepared Scan*

operação nativa. Na Figura 5, é possível verificar a definição do tipo *Scan* em PB.

Tendo toda a informação necessária sobre o `scan` que se pretende efetuar, na fase de execução é possível reutilizar os parâmetros sem que tenham de ser enviados novamente. De todos os parâmetros, o que torna esta solução capaz de lidar com dados estruturados de uma forma mais eficiente é o esquema de colunas pretendido, dado pelo nome `column` na especificação em PB.

Após a obtenção da instância do tipo *Scan* que resulta da junção dos parâmetros recebidos na fase de preparação com os parâmetros recebidos na fase de execução, esta é usada para executar a operação. Uma vez obtidos, os resultados são devolvidos ao cliente no formato demonstrado pela Figura 6. Note que o número de chamadas RPC necessárias para obter os resultados, tal como acontece na operação nativa, depende quer do valor usado para `caching` como do número de linhas a retornar.

Este formato contém um campo, `Last Result`, que informa sobre a existência de mais linhas que satisfazem as condições da operação `scan`, prevenindo a execução de uma nova chamada RPC após os últimos resultados. Seguido deste, existe um conjunto de linhas e cada uma é composta por vários campos: o identificador da linha, um conjunto de índices que indicam a coluna a que cada célula pertence e as células. O conteúdo de cada célula é reduzido apenas aos campos, de acordo com a Figura 1, relativos à marca temporal (`Timestamp`), ao tipo da

```
message Scan {
  repeated Column column = 1;
  repeated NameBytesPair attribute = 2;
  optional bytes start_row = 3;
  optional bytes stop_row = 4;
  optional Filter filter = 5;
  optional TimeRange time_range = 6;
  optional uint32 max_versions = 7 [default = 1];
  optional bool cache_blocks = 8 [default = true];
  optional uint32 batch_size = 9;
  optional uint64 max_result_size = 10;
  optional uint32 store_limit = 11;
  optional uint32 store_offset = 12;
  optional bool load_column_families_on_demand = 13;
  optional bool small = 14;
  optional bool reversed = 15 [default = false];
  optional Consistency consistency = 16 [default = STRONG];
  optional uint32 caching = 17;
}
```

Figura 5: Definição do *Scan* em PB

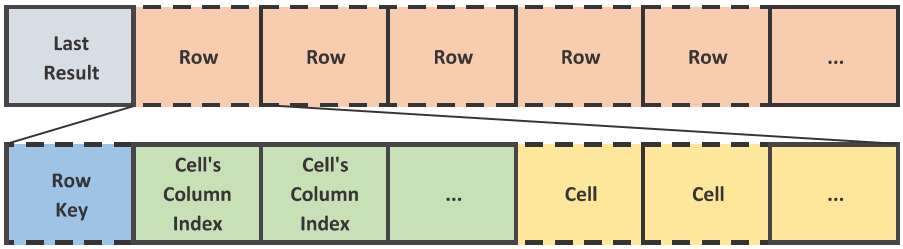


Figura 6: Estrutura dos resultados do *Prepared Scan*

célula (`Type Byte`) e ao valor (`Value`). Note que tanto o identificador de linha, uma vez que está presente no início da linha, como os detalhes das colunas, que são descartáveis devido à presença dos índices, são removidos das células.

3.3 Componente servidor

As chamadas RPC são realizadas a um módulo carregado no *RegionServer* que se denomina de *coprocessor endpoint*. Este é responsável, nesta operação, por atender os pedidos nas várias fases do processo, da seguinte forma:

Preparação

- Insere os parâmetros da instância *Scan*, associados ao cliente, na estrutura partilhada pelas instâncias do *coprocessor*, obtidos através do protocolo;

- Atribui índices às colunas devidamente ordenadas.

Execução
- Atualiza instância *Scan* associada ao cliente com os parâmetros recebidos através do método RPC;
- Obtém os pares chave-valor que satisfazem o *Scan* através da execução de um *InternalScanner* no *RegionServer*, até atingir o número de linhas dado pelo parâmetro `caching` (*default*: 100);
- Por cada linha, remove o identificador de linha em todas as suas células. Por cada célula, atribui o índice da coluna a que pertence e remove os dados relativos ao nome da família de colunas e qualificador;
- Constrói os resultados recorrendo a PB. Caso não tenha atingido o valor dado por `caching`, coloca `true` no campo `Last Result`;
- Retorna resultados ao cliente.

Conclusão Remove a informação relativa ao pedido de preparação e ao cliente.

Note que os parâmetros são guardados numa estrutura de dados partilhada entre as diversas instâncias deste *coprocessor*.

4 Avaliação de desempenho

Para avaliar a solução apresentada ao longo deste documento, recorreu-se à ferramenta de *benchmarking Yahoo! Cloud Serving Benchmark* (YCSB) [17]. Esta ferramenta disponibiliza vários padrões de acesso a dados e permite a personalização de cada um.

4.1 Procedimento

Os *RegionServers* e o *Master* foram alocados em nós distintos, cada um com um processador Intel(R) Core(TM) i3-2100 CPU @ 3.10GHz, 8GB RAM e um disco 7200 RPM. Os *RegionServers*, instanciados com o *coprocessor* do *Prepared Scan*, têm 4GB de *heap* e co-existiram com as respetivas instâncias *DataNode*, assim como o *Master* com o *NameNode*. Neste procedimento, foram usados dois *RegionServers*.

Os testes de desempenho foram executados com vários clientes simulados numa máquina com um processador AMD Opteron(tm) Processor 6172, 120GB RAM e um disco de 7200 RPM. A largura de banda existente entre os clientes e o sistema *Apache HBase* é de 1Gbps.

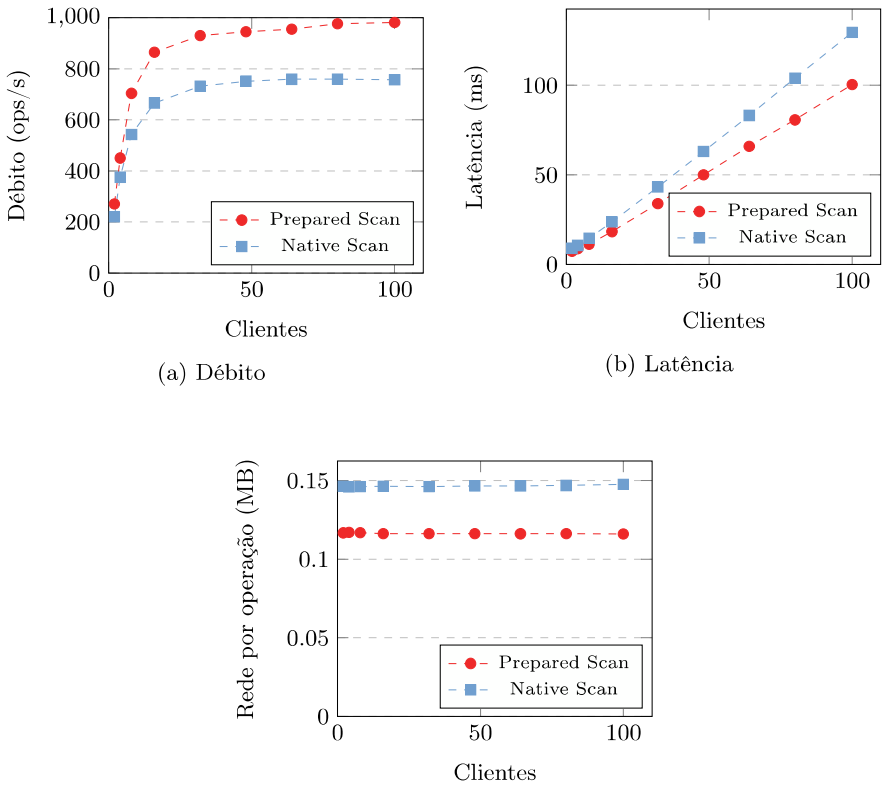
Foi utilizado o *Apache HBase* na versão `1.0.0` e o HDFS na `2.6.0`.

Em termos de dados, povoou-se a base de dados com um milhão de linhas, cada uma com 10 qualificadores, resultando em cerca de dez milhões de pares chave-valor, cujo tamanho do valor é de 100 *bytes*.

Em cada teste, executou-se um milhão de operações usando unicamente a operação `scan` no padrão de acesso a dados, com tamanho fixo de 100 linhas. Uma vez que o *Prepared Scan* é aplicado a um conjunto bem definido de colunas, o cliente YCSB foi modificado de modo a selecionar explicitamente as colunas pretendidas, neste caso todas, no *Scan*, recorrendo ao método `addColumn`. A avaliação foi realizada para 2, 4, 8, 16, 32, 48, 64, 80 e 100 clientes.

4.2 Resultados

Os resultados são demonstrados na Figura 7, divididos em três métricas: débito, visível na Figura 7a, latência, na Figura 7b e, por fim, a rede usada em cada operação, na Figura 7c.



(a) Débito

(b) Latência

(c) Consumo de rede por operação

Figura 7: Métricas de desempenho para *Prepared Scan* e *Scan*

Focando a Figura 7a, é possível constatar que o débito obtido no *Prepared Scan* é sempre superior ao que se obtém por parte da implementação nativa, algo que se evidencia cada vez que o número de clientes é aumentado. Para 2 clientes, a diferença situa-se nas 50 operações por segundo, o que equivale a um aumento de cerca 23%, relativamente ao *Scan* nativo. A diferença no número de operações, nos 80 clientes, apresenta uma diferença de 216 operações por segundo, o equivalente a um acréscimo de cerca 29% no débito. É entre os 80 e os 100 clientes que o *Scan* inicia a sua ligeira quebra no débito, passando de 759

para 756 operações por segundo. Já o débito no *Prepared Scan* se mantém num ligeiro crescimento, atingindo as 981 operações por segundo para 100 clientes.

Em termos de latência, na Figura 7b, verifica-se um afastamento das linhas que representam as duas operações, dando vantagem ao *Prepared Scan*, uma vez que a sua latência aumenta de forma menos acentuada que o *Scan*, à medida que são adicionados mais clientes. No auge do *Scan*, isto é, nos 80 clientes, a diferença entre a latência das duas operações é de 22,41%. Este valor aumenta nos 100 clientes, atingindo os 22,53%.

Um aspeto que era expectável está relacionado com a rede. Previa-se uma redução de dados que se propagam pela mesma, uma vez que a redundância de dados como as colunas e os identificadores de linha é eliminada com o *Prepared Scan*. A Figura 7c representa a métrica de rede utilizada em cada operação e verifica-se efetivamente uma redução de aproximadamente 20% com esta solução.

De acordo com estes resultados, constata-se que o *Prepared Scan* permite lidar com um maior número de clientes e mantendo o desempenho sempre melhor que o *Scan* nativo, quando a estrutura dos dados é conhecida pela aplicação.

5 Conclusão

Neste documento, focou-se a atenção na otimização da consulta de dados estruturados nas bases de dados não relacionais, mais concretamente no *Apache HBase*. Uma das consequências da migração dos dados do contexto relacional para o não-relacional é a perda de conhecimento da estrutura de dados e que tal resulta num processamento e envio de informação extra.

A solução proposta, a adição de uma nova operação *Prepared Scan* diminui o custo associado a essa informação extra através da sua eliminação nos resultados retornados. Com este procedimento, reduziu-se a quantidade de dados enviados para o cliente em cerca de 20% e, consequentemente, foi possível obter um aumento do débito de 29% e diminuir a latência em cerca de 23% com esta solução. O *Prepared Scan* não exige a alteração do código-fonte do *Apache HBase*, uma vez que segue os padrões de extensão de protocolo, isto é, é implementado como um *coprocessor endpoint*. Esta característica permite que esta operação co-exista com a operação nativa e que seja instalada não só em versões mais antigas como também em distribuições diversas, por exemplo *Cloudera*.

Apesar de ser possível obter um melhor desempenho com esta solução, esta pode não ser tão eficiente com uma quantidade de dados na ordem dos *MegaBytes*. Tal se deve à biblioteca de serialização *Protocol Buffers* (PB), que possui algumas limitações já conhecidas pela comunidade do *Apache HBase*[2]. Desta forma, é ainda possível otimizar esta solução recorrendo a métodos fora dos padrões fornecidos pelo sistema.

[2] `https://issues.apache.org/jira/browse/HBASE-7233`

Agradecimentos

Este trabalho foi parcialmente financiado pelo União Europeia no âmbito do projeto CoherentPaaS – Coherent and Rich PaaS with a Common Programming Model – FP7-611068 (`http://CoherentPaaS.eu`).

Referências

1. Rick Cattell. Scalable SQL and NoSQL Data Stores. *SIGMOD Rec.*, 39(4):12–27, May 2011.
2. Lars George. *HBase: The Definitive Guide*. 2011.
3. D. Salmen. Cloud Data Structure Diagramming Techniques and Design Patterns. `https://www.data-tactics-corp.com/index.php/component/jdownloads/finish/22-white-papers/68-cloud-data-structure-diagramming`, 2009.
4. Apache Hive. `http://hive.apache.org`.
5. Apache Phoenix. `http://phoenix.apache.org`.
6. Cloudera Impala. `http://impala.io/`.
7. Julian Rith, Philipp S. Lehmayr, and Klaus Meyer-Wegener. Speaking in Tongues: SQL Access to NoSQL Systems. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 855–857, New York, NY, USA, 2014. ACM.
8. Ricardo Vilaça, Francisco Cruz, José Pereira, and Rui Oliveira. An Effective Scalable SQL Engine for NoSQL Databases. In Jim Dowling and François Taïani, editors, *Distributed Applications and Interoperable Systems*, volume 7891 of *Lecture Notes in Computer Science*, pages 155–168. Springer Berlin Heidelberg, 2013.
9. Liang Lin, Vera Lychagina, Weiran Liu, Younghee Kwon, Sagar Mittal, and Michael Wong. Tenzing: A SQL Implementation On The MapReduce Framework. 2011.
10. A.C. Carniel, A. de Aguiar Sa, V.H.P. Brisighello, M.X. Ribeiro, R. Bueno, R.R. Ciferri, and C.D. de Aguiar Ciferri. Query processing over data warehouse using relational databases and NoSQL. In *Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En*, pages 1–9, Oct 2012.
11. Chongxin Li. Transforming relational database into HBase: A case study. In *Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on*, pages 683–687, July 2010.
12. Dhruba Borthakur, Jonathan Gray, Joydeep Sen Sarma, Kannan Muthukkaruppan, Nicolas Spiegelberg, Hairong Kuang, Karthik Ranganathan, Dmytro Molkov, Aravind Menon, Samuel Rash, et al. Apache Hadoop Goes Realtime at Facebook. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1071–1080. ACM, 2011.
13. Apache HBase. `http://hbase.apache.org`.
14. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A Distributed Storage System for Structured Data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.
15. Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

16. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.
17. Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

Exclusão Mútua Tolerante a Faltas Bizantinas na Cloud

Ricardo Mendes, Tiago Oliveira, Alysson Bessani

LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal
{rmendes,toliveira}@lasige.di.fc.ul.pt,anbessani@fc.ul.pt

Resumo Recentemente, uma série de sistemas têm sido propostos para o armazenamento e partilha de dados em múltiplos serviços de cloud. Estes trabalhos usualmente requerem o uso de algoritmos de exclusão mútua para permitir aos clientes reservar recursos por determinados períodos de tempo com vista a evitar o uso concorrente desses recursos. Neste artigo apresentamos um novo protocolo de exclusão mútua que tira partido de vários objetos de trinco implementados em diferentes serviços da cloud para garantir o correcto funcionamento do protocolo mesmo na presença de provedores de cloud faltosos. Este novo protocolo é *centrado em dados*, não necessitando portanto de nenhum servidor específico a correr nas clouds para além dos serviços propriamente ditos. Outra contribuição deste trabalho é a implementação e descrição de vários objetos de trinco que usam diferentes serviços de cloud no mercado. Apresentamos ainda um estudo dos custos inerentes ao uso tanto do novo protocolo de exclusão mútua como das referidas implementações de objetos, assim como uma avaliação de desempenho dos mesmos em diferentes contextos de utilização.

Palavras-chave: exclusão mútua, coordenação, computação em clouds, tolerância a faltas bizantinas.

1 Introdução

Nos últimos anos têm surgido um conjunto de sistemas que propõem o uso de múltiplos serviços de armazenamento na cloud (e.g., Amazon S3, Microsoft Azure Storage, Google Storage, Rackspace Files) para armazenar e partilhar dados [9,11,12,16]. Nestes sistemas, os dados a serem armazenados na cloud são replicados por múltiplos serviços - chamados *cloud-of-clouds* - de tal forma que os dados possam sobreviver a falhas nos serviços de clouds. A maioria desses sistemas consideram um modelo de dados em que cada item de dados (ou ficheiro) pode ser escrito por apenas um utilizador de cada vez, i.e., não suportam escritas concorrentes num mesmo item de dados.

A fim de garantir que dois escritores nunca tentam escrever num mesmo item de dados, esses algoritmos assumem a existência de um algoritmo de exclusão mútua. Alguns sistemas como o RACS [9] e o SCFS [12] usam serviços de coordenação (e.g., Zookeeper [19]) para concretizar essa primitiva. Esta primitiva permite a partilha controlada de ficheiros entre os clientes destes sistemas.

Outros sistemas, como é o caso do DepSky [11], incluem algoritmos de exclusão mútua desenhados especificamente para um ambiente de multiplas clouds. A grande vantagem destes algoritmos em relação ao uso de sistemas de coordenação reside no fato de eles não requererem a execução de servidores, sendo executados directamente sobre as clouds de armazenamento. Por outro lado, um dos seus problemas fundamentais prende-se com o seu baixo desempenho: para obter acesso à zona crítica, são necessários três acessos à cloud, mesmo em execuções sem acessos concorrentes.

Neste artigo apresentamos um novo algoritmo centrado em dados para a cloud-of-clouds. Diferentemente do algoritmo proposto no DepSky, o nosso algoritmo considera o uso de diferentes tipos de serviços de cloud (e.g., Amazon DynamoDB, Azure Queue, Google Datastore), ao invés de apenas serviços de armazenamento de objetos (e.g., Amazon S3). Além disso, o algoritmo é construído de forma modular: inicialmente construímos algoritmos de exclusão mútua base em cima de cada serviço de cloud que usamos, e depois combinamos estes algoritmos numa versão composta que tolera faltas bizantinas em até um terço dos serviços empregados. Desta forma, o algoritmo proposto apresenta um desempenho muito melhor do que o usado no DepSky.

Outra vantagem importante do nosso algoritmo é o fato de ter poucos requisitos de sincronia. A maioria dos algoritmos de exclusão mútua e de gestão de trincos na presença de processos faltosos requer hipóteses temporais para garantir a *safety* do sistema. O nosso algoritmo requer tais hipóteses apenas para garantir *liveness*, tornando-o muito mais apropriado para ser usado na internet.

Em sumário, este artigo apresenta as seguintes contribuições:

1. um conjunto de algoritmos de exclusão mútua eficientes construídos sobre diferentes serviços de cloud;
2. um algoritmo de exclusão mutua tolerante a faltas bizantinas que faz uso de serviços disponíveis na internet;
3. as provas de correção de todos os algoritmos mencionados anteriormente e uma avaliação de custo e desempenho dos mesmos tendo em conta vários serviços de cloud reais.

Os algoritmos apresentados neste artigo estão a ser usados no sistema de ficheiros distribuído Charon, usado para integrar biobancos e bioinformáticos no contexto do projeto BiobankCloud [13].

2 Trincos na *Cloud-of-Clouds*

Locks são contratos usados para controlar acessos concorrentes a recursos (e.g. um ficheiro), prevenindo assim conflitos de versões. *Leases* [17], tal como os *locks*, são também contratos usados para coordenar o acesso a recursos partilhados, mas com uma noção de tempo de validade que permite que um recurso reservado por um cliente volte a ficar disponível em caso de falha deste após esse período de validade expirar. Doravante neste artigo, adoptamos esta última definição para o termo português *trinco*.

Sistemas como o RACS [9] e o SCFS [12] coordenam acessos concorrentes a recursos recorrendo a algoritmos de exclusão mútua feitos em cima de serviços de coordenação [19] específicos instalados em VMs nas clouds. Por sua vez, os algoritmos de exclusão mútua centrados em dados e tolerantes a faltas existentes (e.g., [11]) são desenhados para funcionar em cima de serviços de armazenamento de objetos. O problema destas soluções são, no primeiro caso os custos de manter as VMs, e em ambos a sua falta de modularidade. Neste artigo nós propomos um algoritmo centrado em dados e tolerante a faltas bizantinas mas modular pois permite tirar partido de qualquer serviço fornecido pelas clouds (não só o de armazenamento). Na prática, utilizamos um protocolo que utiliza $3f + 1$ objetos que permitem a gestão de trincos (aos quais chamamos *objetos de trinco base*) para implementar um objecto (*objecto de trinco composto*) que tolera f faltas por parte dos serviços de cloud utilizados pelos objetos base. Esta capacidade é importante pois, desta forma, é possível usar serviços de cloud com melhor desempenho ou com suporte a funções com maior poder de sincronização [18] para melhorar o desempenho do algoritmo.

O algoritmo de trinco do DepSky tem ainda a limitação de necessitar de sincronização dos relógios entre os clientes. A nossa solução não tem esta limitação pois, ao invés de utilizar valores dos relógios dos clientes utiliza os das clouds (§ 4).

2.1 Modelo e Garantias do Sistema

O nosso modelo de sistema é equivalente a outros modelos usados nos algoritmos tradicionais centrados em dados e tolerantes a faltas bizantinas (e.g., [8,11]). Um número ilimitado de clientes podem aceder a um conjunto de objetos base (e.g., serviços de cloud) que compreendem o serviço de exclusão mútua. Todos esses clientes, e no máximo f objetos base, podem sofrer faltas bizantinas. Cada um dos objetos base fornece mecanismos de controlo de acesso, de forma a ser garantido que somente clientes autorizados podem invocar operações sobre eles [21]. Uma vez que a noção de trinco implica garantias de tempo, é assumido um limite máximo de tempo para a transmissão de mensagens entre os clientes e os objetos base. Contudo, *esta assumpção é só necessária para a propriedade de liveness.*

O nosso algoritmo garante exclusão mútua (sempre segura) pois garante que, em cada momento, está no máximo um processo a aceder ao recurso partilhado e que esse processo só vai aceder ao recurso numa quantidade de tempo limitada. Cada recurso fornece três operações de trinco diferentes: `obter(T)`, `renovar(T)` e `libertar()`. As operações `obter(T)` e `renovar(T)` têm a função de adquirir e renovar o trinco por T segundos, respectivamente. Por sua vez, a operação `libertar()` serve para terminar o trinco. Estas operações satisfazem as seguintes propriedades:

- *Exclusão Mútua (safety):* Nunca existem dois clientes correctos com um trinco válido em simultâneo.
- *Liberdade de Obstrução (liveness):* Um cliente correcto que tente adquirir o trinco sem concorrência terá sucesso.
- *Limitação Temporal (liveness):* Um cliente correcto que obtém o trinco vai possui-lo por um máximo de T unidades de tempo, excepto se este for renovado.

Estas propriedades não impedem um cliente bizantino de adquirir o trinco e renová-lo indefinidamente, nem impedem o acesso direto ao recurso sem um trinco válido. No entanto, isto é aceitável pois um cliente malicioso pode sempre danificar todos os recursos (e.g., ficheiros) aos quais tenha acesso. Do mesmo modo, se um cliente falha enquanto possui o trinco, este estará disponível novamente passados, no máximo, T unidades de tempo (propriedade de *limitação temporal*).

O nosso algoritmo não satisfaz nenhuma propriedade mais forte que as descritas acima pois isso implicaria um número superior de acessos às clouds, resultando numa diminuição do desempenho do algoritmo. Esta escolha justifica-se com o facto de a concorrência esperada entre clientes, no domínio dos sistemas de partilha de dados em cloud reais, ser reduzida.

3 Protocolo de Trinco Composto Tolerante a faltas Bizantinas

A operação *obter()* do objecto de trinco composto é apresentada no Algoritmo 1. Com vista a obter um trinco composto, um cliente chama simultaneamente a operação *obter()* em todos os $3f + 1$ objetos de trinco base (linhas 5–6) e espera, ou por $2f + 1$ repostas bem sucedidas, ou por $f + 1$ respostas de operações que falharam (linha 7). Note que só no primeiro caso o cliente obtém o trinco. Caso contrário o trinco está indisponível ou sob concorrência e, desta forma, o cliente necessita de libertar todos os trincos que potencialmente tenha obtido – tanto os advidos de operações bem sucedidas como os das operações para as quais ainda não foi obtida uma resposta (linhas 11–12). Neste caso, o cliente espera um pouco, repetindo o processo descrito após algum tempo (linha 13). Este algoritmo é repetido até ser obtido sucesso na aquisição do trinco composto ou até um temporizador expirar.

ALGORITMO 1: Trinco composto pelo cliente c.

```
1  function obter(time) begin
2      res ← false;
3      repeat
4          L[0 .. n − 1] ⟵ ⊥;
5          parallel for 0 ≤ i ≤ n − 1 do
6              L[i] ⟵ trincoBase_i.obter(time);
7          wait until i : (|{L[i] = true}| > 2f) ∨ (|{L[i] = false}| > f);
8          if |{i : L[i] = true}| > 2f then
9              res ← true;
10         else
11             for i : (L[i] = ⊥) ∨ (L[i] = true) do
12                 trincoBase_i.libertar();
13             dormir por algum tempo;
14     until res ≠ false;
15     return res;
```

Para libertar o trinco composto basta libertar $n-f$ trincos base. Na renovação de um trinco o processo é similar ao da operação de *obter()* mas é necessário um

acesso adicional às clouds para remover a informação relacionada com o trinco a ser renovado.

3.1 Prova de correção do Protocolo de Trinco Composto

Nesta secção apresentamos a prova de correção do protocolo de trinco composto tolerante a faltas bizantinas. Com vista a provar a exclusão mútua no acesso a recursos é necessário definir com precisão o que significa para um processo deter um trinco.

Definição 1 *Um cliente correcto c detém um trinco no instante t por $T' > 0$ unidades de tempo se obtiver T' como resposta da execução da operação obter(T) num quorum de objetos de trinco base.*

Dada esta definição, vamos proceder à prova das propriedades do Algoritmo 1. Na prova destas propriedades usamos a função C() que mapeia o valor do relógio local de um processo para o valor de um relógio de tempo real (global). Note-se que os processos não têm acesso a esta função, sendo que esta é apenas um instrumento teórico para podermos definir a correção do protocolo.

Teorema 1 (Exclusão Mútua). *Nunca existem dois clientes correctos com um trinco válido em simultâneo.*

Prova. Assuma que isto é falso: existe um instante de tempo real t no qual dois clientes c_1, c_2 detêm o trinco. Vamos provar que esta assumpção leva a uma contradição. Seja t_1 (resp. t_2) o instante do relógio local no qual a operação *obter(T)* retorna o valor T' ao cliente c_1 (resp. c_2), i.e., o momento no qual o cliente obtém o trinco. Seja também t_{start1} (resp. t_{start2}) o instante do relógio local no qual a operação *obter(T)* é chamada por c_1 (resp. c_2). Dado isto, o momento no qual c_1 assume que o trinco expirou é $v_1 = t_{start1} + T'$ (resp. c_2 and $v_2 = t_{start2} + T'$). Com estas definições, a existência de t requer $\mathrm{C}(t_2) \leq \mathrm{C}(v_1)$ ou $\mathrm{C}(t_1) \leq \mathrm{C}(v_2)$.

Tendo em conta que a invocação de *obter(T)* precede o seu retorno, e que este precede a expiração do trinco em todos os objetos de trinco base, nós temos:

$$\mathrm{C}(t_{start1}) < \mathrm{C}(t_1) < \mathrm{C}(v_1) \tag{1}$$

$$\mathrm{C}(t_{start2}) < \mathrm{C}(t_2) < \mathrm{C}(v_2) \tag{2}$$

Assuma o caso esquerdo apresentado na figura 1. Neste caso, visto cada objecto de trinco base garantir a propriedade de *exclusão mútua*, o cliente c_2 só estará apto a obter o trinco quando o trinco detido por c_1 tiver expirado em pelo menos $n - f$ objetos base, i.e, $\mathrm{C}(t_1) + T' < \mathrm{C}(t_2)$. Dada esta condição juntamente com as equações 1 e 2 nós temos:

$$\begin{aligned} \mathrm{C}(t_1) + T' < \mathrm{C}(t_2) \implies \mathrm{C}(t_{start1}) + T' < \mathrm{C}(t_2) \implies \\ (\mathrm{C}(t_{start1} + T') < \mathrm{C}(t_2) \implies \mathrm{C}(v_1) < \mathrm{C}(t_2)) \end{aligned} \tag{3}$$

A equação 3 contradiz o caso da esquerda da figura 1 para a existência de t. A mesma abordagem tem de ser usada, assumindo que c_2 obtém o trinco antes de c_1, para provar que a condição associada a esta situação é também impossível.

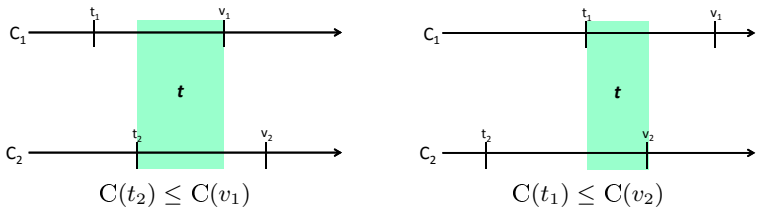


Fig. 1. Ilustração da prova de exclusão mútua.

Teorema 2 (Liberdade de Obstrução). *Um cliente correcto que tente adquirir o trinco sem concorrência terá sucesso.*

Prova. O cliente c começa por tentar obter $n - f$ respostas de sucesso (trincos) dos objetos base (linhas 5 e 6). Ele irá obter o trinco em todos os objetos base correctos pois, (1) nenhum outro cliente detém o trinco, (2) nenhum outro cliente está a tentar obter obter o trinco concorrentemente, e (3) cada objecto de trinco base tem (por definição) de obrigatoriamente garantir a propriedade de "*liberdade de obstrução*". O cliente c irá então obter o trinco do object composto após obter $n - f$ respostas bem sucedidas dos objetos base.

Teorema 3 (*Limitação Temporal*). *Um cliente correcto que obtém o trinco vai possui-lo por um máximo de T unidades de tempo, excepto se este for renovado.*

Prova. Vamos provar este teorema por contradição. Assumamos então que o a definição do mesmo é falsa, i.e., existe um caso no qual um cliente c detém o trinco por $T' > T$ unidades de tempo. Sejam t_1 e t_{start} os instante do relógio local no qual a operação *obter(T)* retorna o valor T' a c_1, ou seja, o momento no qual o cliente obtém o trinco e o momento em que esta operação é chamada pelo mesmo cliente, respectivamente.

Com isto, c_1 detém o trinco por $T' > T$ unidades de tempo e, dado que a invocação da operação *obter(T)* precede o seu seu retorno ($t_{start} < t_1$), temos:

$$T' = T - (t_1 - t_{start}) > T \implies -(t_1 - t_{start}) > 0 \implies \\ (t_1 - t_{start} < 0 \implies t_1 < t_{start}) \tag{4}$$

O resultado da equação 4 contradiz a definição causal de t_{start} e de t_1.

4 Implementação de Trincos Base

A maioria dos serviços fornecidos pelas *clouds* públicas, desde o armazenamento de objetos até às bases de dados atómicas, fornecem mecanismos que permitem a criação de objetos de trinco base. No entanto, todas as implementações de objetos base têm de ser condizentes com algumas especificações de forma a cooperarem entre si como peças integrantes de um trinco composto.

Primeiramente, a operação `obter()` requer a criação bem sucedida de diferentes tipos de entradas de trinco no serviço de cloud. Em segundo lugar, os clientes são responsáveis por colectar as suas entradas de trinco que estejam expiradas

ou inválidas com vista a poupar recursos. Na maioria das implementações, esta limpeza requer pelo menos uma operação extra à cloud nas operações de *obter()* e de *renovar()*. Terceiro, as entradas de trinco têm de ser assinadas antes de serem enviadas para a cloud para garantir que os provedores de cloud não conseguem criar ou corromper trincos. Em quarto lugar, as implementações dos objetos trinco base usam os mecanismos de controlo de acesso dos serviços de cloud para garantir que apenas clientes autorizados podem aceder ao trinco. Desta forma, clientes maliciosos só conseguem prejudicar clientes correctos se estes inadvertidamente lhes tiverem dado acesso ao recurso. Quinto, os clientes não usam o seu relógio local para marcar as entradas de trinco (para marcar o início do seu período de validade). Ao invés disso, eles contam com o serviço de cloud para marcar as entradas de trinco ou usam os valores do relógio da cloud retornados de cada operação efectuada aos serviços. Estas marcas são também usadas para verificar a validade do trinco (ao invés do relógio local).

Nesta secção vamos descrever quatro implementações de objetos de trinco base desenvolvidos em cima de diferentes serviços oferecidos por alguns dos mais populares provedores de cloud. Embora estas implementações utilizem diferentes serviços na cloud, todas elas foram desenhadas seguindo um conjunto similar de técnicas que nos permitem lidar com comportamentos maliciosos e a duração dos trincos.

4.1 Armazenamento de objetos

Os serviços de armazenamento de objetos mantêm objetos de dados de tamanho variável em contentores que são acessíveis através de uma interface de armazenamento de pares chave-valor hierárquica. O algoritmo 2 apresenta a nossa implementação de um objecto de trinco base para este serviço de cloud.

Este algoritmo funciona em três passos. Um cliente começa por listar todos os objetos no contentor (linha 3). Se nenhuma entrada de trinco válida for encontrada ou se for encontrada uma entrada que pertença ao próprio cliente (um trinco que necessita ser renovado), significa que o trinco está livre e o cliente pode tentar obtê-lo (linhas 7-10 e 13), caso contrário o trinco não pode ser obtido e a função retorna *false* (linha 12). Para obter o trinco o cliente chama o procedimento *performLease*() no qual, primeiramente, ele insere uma nova entrada assinada no contentor (linha 16), e de seguida lista os objetos novamente para verificar se nenhuma outra entrada de trinco foi inserida concorrentemente por outro cliente (linha 17). Se nesta segunda listagem for observada outra entrada de trinco válida, o cliente remove a sua entrada de trinco e retorna *false* (linhas 18-20). Caso contrário, e antes de retornar o sucesso da operação, é necessário perceber se esta corresponde à renovação de um trinco. Nesse caso, a entrada de trinco mais antiga é removida (linhas 21 e 22).

Para libertar um trinco, o cliente tem apenas de remover a sua entrada de trinco do serviço de armazenamento.

Este algoritmo funciona em serviços como o Amazon S3 [2], Google Storage [4], Azure Blob Storage [14] e o Rackspace Files [7] visto todos eles garantirem consistência forte no que diz respeito à criação de objetos e fornecerem o valor do relógio do respectivo provedor de cloud em todas as respostas às operações em si efectuadas.

ALGORITMO 2: Trinco no serviço de armazenamento pelo cliente c.

```
1  function servArmazenamentoObter(time)
2  begin
3      L ⟵ cloud.list();
4      cTime ⟵ cloud.getTime();
5      lease_id ←"lease-"+c+"-"+time;
6      index ⟵ 0;
7      foreach lease-c'-T'_i ∈ L do
8          if lastTimeModified(lease-c'-T'_i) + T' > cTime ∧ verify(lease-c'-T'_i, K_{u_c'})) then
9              if c' = c then
                   // é uma operação de renovação
10                 return performLease(lease_id,lease-c'-T'_i);
11             else
12                 return false;
13     return performLease(lease_id, ⊥);
14 procedure performLease(lease_id, oldLease)
15 begin
16     cloud.write(lease_id, sign(lease_id, K_rc));
17     L ⟵ cloud.list();
18     if ∃e ∈ L : e ≡lease-c'-T' : c' ≠ c ∧ lastTimeModified(e)+T'> cTime ∧ verify(e, K_{u_c'}))
       then
19         cloud.delete(lease_id));
20         return false;
       // verificar se é uma renovação
21     if oldLease ≠⊥ ∧oldLease ≠ lease_id then
22         cloud.delete(oldLease);
23     return true;
```

Prova informal. Para provar que este algoritmo cumpre a propriedade de exclusão mútua é necessário perceber que, para que haja dois clientes com um trinco válido em simultâneo, ambos teriam que ver a sua entrada de trinco como sendo a única válida na segunda listagem (linha 17) num determinado período. Claramente isto é impossível pois, visto que estes escrevem a sua entrada de trinco antes da segunda listagem, ambos verem apenas a sua entrada de trinco significaria que as suas visões do serviço eram inconsistentes, o que é impossível dado que estes serviços são fortemente consistentes na criação das entradas de trinco. Na mesma linha, provamos que a propriedade de liberdade de obstrução é garantida pois, se um cliente estiver a executar o algoritmo sem concorrência, claramente a sua entrada de trinco vai ser a única válida devolvida na segunda listagem. O algoritmo garante também a propriedade de limitação temporal pois a validade das entradas de trinco é testada sempre que o algoritmo é executado e por todos os clientes (linha 8), comparando o momento de criação do trinco (*lastTimeModified*()) com o valor actual do relógio da cloud. Caso a entrada de trinco já tenha passado o seu período de validade, significa que o cliente já não detém o trinco.

4.2 Filas Aumentadas

Serviços como a Windows Azure Queue [6] e a Rackspace Queue [5] têm operações de *enqueue* (inserir), *dequeue* (obter e remover) e *list* (listar) fortemente consistentes, fornecendo desta forma uma abstracção de memória partilhada universal capaz de resolver problemas de sincronização [18]. Uma característica importante

fornecida por estes serviços de fila é o facto de ambos permitirem ao cliente especificar a duração máxima na qual os dados (no nosso caso as entradas de trinco) estão válidos no serviço. Após esse período de tempo, a entrada desaparece da fila. O algoritmo 3 apresenta a nossa implementação de objetos de trinco base utilizando estes serviços.

No algoritmo, o cliente começa por listar os objetos presentes na fila para verificar se existem entradas de trinco de outros clientes (linha 4). Se no resultado da listagem não existir nenhuma entrada de trinco de um outro cliente, significa que ele pode tentar obter o trinco (através da chamada do procedimento *performLease*() (linhas 5 e 6), caso contrário o trinco não pode ser obtido e é retornado *false* (linha 7). Neste ponto o cliente insere uma nova entrada de trinco assinada e lista novamente todas as entradas da fila para verificar se a sua entrada é a entrada válida com menor índice (cabeça da fila), o que significa que o cliente detém o trinco (linhas 12-14). Neste momento, o cliente remove todas as entradas da fila com a excepção da sua (linhas 14-16), tanto para remover as suas entradas de trinco desactualizadas (na operação *renovar()*), como para garantir que nenhum outro cliente obtém o trinco inadvertidamente quando a sua entrada expirar e desaparecer da fila. Para libertar um trinco neste objecto base, o cliente apenas necessita remover a sua entrada de trinco da fila.

Prova informal. O algoritmo cumpre a propriedade de exclusão mútua pois, para que hajam dois clientes com o trinco em simultâneo, teriam que existir dois trincos válidos com o menor índice na fila na segunda listagem (linha 11), o que é claramente impossível pois, independentemente da ordem pela qual os pedidos de inserção de entradas de trinco sejam executados pelo serviço, apenas um terá o menor índice. Garante também a propriedade de liberdade de obstrução pois, se um cliente estiver a executar o algoritmo sem concorrência, após a inserção

ALGORITMO 3: Trinco em filas aumentadas pelo cliente c.

```
1  function filasObter(T)
2  begin
3      lease_id ⟵ "lease-"+c+nonce+"-"+sign(lease_id, K_{r_c});
4      L ⟵ queue.list();
5      if ∄e ∈ L : e ≡lease-c'-nonce'-s ∧ c' ≠ c ∧ verify(s, K_{u_{c'}}) then
6          return performLease(L, lease_id, T);
7      return false;
8  procedure performLease(L, lease_id, T)
9  begin
10     queue.add(lease_id, T);
11     L_2 ⟵ queue.list();
12     for 0 ≤ i ≤ size(L_2) do
13         if L_2[i] ≡lease-c'-nonce'-s ∧ verify(s, K_{u_{c'}}) then
14             if L_2[i] = lease_id then
15                 queue.delete(L);
16                 return true;
17             else if c' ≠ c then
18                 queue.delete(lease_id);
19                 return false;
20     return false;
```

da entrada de trinco, esta será a única válida logo a que tem menor índice. A propriedade de limitação temporal é também garantida pois os serviços de fila garantem que a entrada de trinco só fica na fila durante o período pré-definido (que define a validade do mesmo).

4.3 Bases de dados NoSQL

O Amazon DynamoDB [1] é um serviço fortemente consistente que armazena dados como pares que contêm uma chave única associada com um conjunto de valores, e que fornece uma operação de actualização que permite a implementação de objetos base eficientes. No algoritmo 4 é apresentada a implementação de um objecto de trinco base que tira partido deste serviço.

Neste algoritmo, o cliente verifica se já existe alguma entrada de trinco válida na base de dados. Em caso afirmativo, e se esta entrada pertencer a outro cliente a operação retorna *false* (linhas 3–7). Caso contrário, o cliente escreve a nova entrada de trinco assinada usando a operação de actualização condicional (linha 12). Esta operação garante que a entrada é actualizada apenas no caso de nenhum outro cliente ter adicionado uma entrada de trinco no entretanto, retornando no resultado se a actualização teve ou não sucesso.

Para libertar o trinco, basta remover a entrada de trinco da base de dados.

Prova informal. Para um cliente obter um trinco tem de obter sucesso da operação *testAndSetItem*() (linha 12). Este algoritmo garante exclusão mútua pois, visto esta operação ser atómica, se ambos executarem esta operação concorrentemente, independentemente do valor do parâmetro *res*, apenas um irá ter sucesso. Da mesma forma, este garante também liberdade de obstrução pois, visto não haver concorrência, o cliente irá ter $res = \perp$ e a operação *testAndSetItem*() irá ser bem sucedida. Garante ainda a propriedade de limitação temporal pois a validade dos trincos é testada a cada execução do algoritmo por todos os clientes (linha 6), comparando o momento de criação do trinco com o valor actual (usando o relógio da cloud).

ALGORITMO 4: Trinco em bases de dados NoSQL pelo cliente c.

```
1  function bdNoSlqObter(time)
2  begin
3      res ⟵ db.query("key", EQ,"lease");
4      cloudTime ⟵ db.getTime();
5      if res ≠⊥ ∧res.cId ≠ c then
6          if res.expirationTime < cloudTime ∧ verify(res, K_{u_res.cId}) then
7              return false;
8      lease.key ⟵"lease";
9      lease.expirationTime ⟵ cloudTime + time;
10     lease.cId ⟵ c;
11     lease.sign ⟵ sign(lease, K_{r_c});
       // apenas é bem sucedida se o valor no serviço for igual ao valor dado (i.e. res)
12     succeed ⟵ db.testAndSetItem(res, lease);
13     return succeed;
```

4.4 Bases de dados transaccionais

Este tipo de serviços armazena os dados em tabelas e suportam transacções ACID. Um exemplo é o Google Datastore [3], que é uma base de dados fornecida como serviço de cloud. Contrariamente ao Amazon DynamoDB, este não suporta a função de *test-and-set*, no entanto suporta transacções atómicas que permitem, da mesma forma, a implementação eficiente de objetos de trinco base. O algoritmo 5 mostra essa implementação.

A operação de *obter()* é executada numa transacção. O cliente começa por iniciar a transacção e efectuar uma pesquisa na base de dados por uma entrada de trinco (linhas 3 e 4). Se existir uma entrada válida que pertença a outro cliente, a transacção é abortada e a operação retorna *false* (linhas 8-10). Caso contrário, o cliente está apto a tentar obter/renovar o trinco através da chamada do procedimento *performLease*(). Neste ponto, consoante a operação seja uma renovação ou uma obtenção de um trinco, o cliente chama as operações *update* ou *insert* do serviço de cloud, respectivamente, para actualizar/inserir uma entrada de trinco assinada (linhas 18–21). Depois disto, é feito o *commit* da transacção, dependendo o sucesso na obtenção do trinco do sucesso desta operação (linha 22).

A libertação do trinco, à imagem de outros algoritmos apresentados, é feita pela remoção da entrada de trinco da base de dados.

Prova informal. Este algoritmo garante exclusão mútua pois, visto a transacção ser iniciada antes da execução da operação de pesquisa (linha 3), e o *commit* da transacção ser executado após a inserção/actualização da entrada de trinco, estas operações são feitas de forma atómica. Isto garante que, entre a operação de pesquisa e o *commit*, não existe concorrência de operações entre clientes. Assim,

ALGORITMO 5: Trinco em bases de dados transaccionais pelo cliente c.

```
1  function bdTransObter(time)
2  begin
3      trans ⟵ ds.beginTransaction();
4      res ⟵ ds.lookUp("lease", trans);
5      cloudTime ⟵ ds.getTime();
6      if res =⊥ then
7          return performLease((cloudTime + time), false, trans);
8      if res.cId ≠ c ∧ res.expirationTime < cloudTime ∧ verify(res.sign, K_{u_{res.cId}}) then
9          ds.abort(trans);
10         return false;
11     return performLease((cloudTime + time), true, trans);
12 procedure performLease(expTime, isRenew, trans)
13 begin
14     lease.key ⟵"lease";
15     lease.cId ⟵ c;
16     lease.expirationTime ⟵ expTime;
17     lease.sign ⟵ sign(lease, K_{r_c});
18     if !isRenew then
           // apenas é bem sucedida se não existir uma chave igual no serviço
19         ds.insert(lease, trans);
20     else
21         ds.update(lease, trans);
22     return ds.commit(trans);
```

se dois clientes tentarem obter o trinco concorrentemente apenas um deles terá oportunidade de inserir uma entrada no serviço e assim obter o trinco, pois o outro irá obter uma entrada de trinco válida na operação de pesquisa. A liberdade de obstrução é garantida pois, se um cliente estiver a executar o algoritmo sem concorrência, não existirá nenhuma entrada válida na operação de pesquisa logo este irá inserir uma nova, e obter o trinco. O algoritmo garante a limitação temporal pois, tal como em algoritmos anteriores, a validade das entradas de trinco é testada a cada execução do algoritmo (linha 8) utilizando o relógio da cloud.

4.5 Comparação de Objetos de Trinco Base

A tabela 1 resume as propriedades dos diferentes objetos de trinco base implementados em cima de diferentes serviços de cloud.

Esta tabela mostra que a maioria dos objetos base necessitam de três acessos às clouds para implementa a operação *obter()*. A liberdade de obstrução é a propriedade de progresso suportada por todos os algoritmos, no entanto, todos os algoritmos que não são baseados em serviços de armazenamento de objetos satisfazem também a propriedade de *liberdade de impasse* [10]. A propriedade de *liberdade de impasse* garante que se dois ou mais clientes tentarem obter o trinco de forma concorrente, um deles é bem sucedido. Actualmente,o nosso protocolo de trinco composto satisfaz apenas a propriedade de liberdade de obstrução, mesmo que os n objetos de trinco base garantam a propriedade de *liberdade de impasse*.

Há uma diferença de custos significativa em executar os algoritmos de trinco base. No entanto, o nosso trinco composto vai ser significativamente mais barato que correr um serviço de trincos tolerante a faltas em múltiplas VMs nas clouds, especialmente se considerarmos que os trincos são apenas necessários para coordenar escritas e são mantidos por algum tempo. Mais especificamente, cada obtenção de trinco custará cerca de μ\$40 enquanto que ter VMs em quatro provedores diferentes puderá custar \$1200 por mês, mais o custo com o tráfego de dados (que é significante pois as réplicas do serviço de trincos precisam de se sincronizar em cada operação de obtenção de trinco) [12] e o custo com manutenção.

Serviço	Acessos	Custos (μ\$)	Progresso
Amazon S3	3	15	Liberdade Obst.
Google Storage	3	30	Liberdade Obst.
Azure Blob Storage	3	0.108	Liberdade Obst.
RackSpace Files	3	8.640	Liberdade Obst.
Azure Queue	3	0.15	Liberdade Impasse
RackSpace Queue	3	30.144	Liberdade Impasse
Amazon DynamoDB	2	subscrição	Liberdade Impasse
Google Datastore	4	1.2	Liberdade Impasse

Tabela 1. Objetos de trinco base criados em cima de serviços de cloud. A tabela mostra o número de acessos necessários para obter o trinco na ausência de concorrência; os custos monetários (em micro-dólares) dessa operação; a propriedade de progresso satisfeita por cada objecto base, sejam *liberdade de obstrução* ou liberdade de impasse (que é mais forte). *Este serviço é pago popr subscrição mensal.

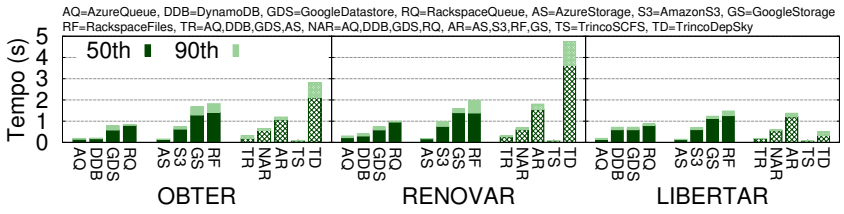


Fig. 2. Latência das operações de obtenção, renovação e libertação do trinco (sem concorrência) para todos os objetos base, para várias configurações do trinco composto, e para dois dois algoritmos de outros trabalhos, nomeadamente o do DepSky e do SCFS.

5 Avaliação

Nesta secção avaliamos o trinco composto e todas as implementações de de objetos base com e sem concorrência. Comparamos ainda dos algoritmos que apresentamos com os algoritmos de exclusão mútua do DepSky e do SCFS. O DepSky foi configurado para utilizar os serviços: Amazon S3 (US), Google Storage(US), Azure Storage (UK) e Rackspace Files (UK). Também o serviço de coordenação do SCFS foi instalado em VMs *standard* no Amazon EC2 (EU), no Microsoft Azure (EU), na Rackspace Servers (EU) e na ElasticHosts (EU).

5.1 Execuções sem concorrência

A Figura 2 apresenta a latência para obtenção, renovação e libertação do trinco na ausência de concorrência para várias configurações do algoritmo de trinco composto, para os objetos de trinco base e para o algoritmo de exclusão mútua do DepSky [11] (TD) e do SCFS [12] (TS). Note-se que no que diz respeito às configurações compostas, na composição definida por TR são aglomerados os 4 objetos de trinco base mais rápidos, mesmo que estejam no mesmo provedor de cloud. A composição NAR reúne os objetos base de serviços que não são de armazenamento, ao passo que AR mostra resultados para algoritmo composto usando só objetos base que usam serviços de armazenamento.

Os objetos de trinco base baseados nos serviços que não são armazenamento necessitam de um tempo entre 200 ms a 1.8 s para obter o trinco. Por sua vez, os objetos de trinco base referentes a serviços de armazenamento obtêm o trinco num tempo compreendido entre 200 ms e 2.8 s. No geral, as latências apresentadas por serviços de armazenamento são piores que as dos restantes serviços. Nós pensamos que isto acontece porque os serviços de armazenamento em cloud são orientados á taxa de transferência, sendo menos eficazes quando lidam com objetos pequenos (entradas de trinco têm aproximadamente 60 bytes).

Os resultados para a obtenção do trinco nas configurações de trinco composto reflectem o desempenho dos seus objetos de trinco base. Mais especificamente, o protocolo de trinco composto espera por um quorum de $2f + 1 = 3$ confirmações dos seus objetos base, o que significa que a latência do objecto composto é similar à latência do terceiro objecto base mais rápido. Um exemplo disto é a configuração NAR cuja latência é similar ao GDS, que é pior que o DDB e AQ, mas melhor

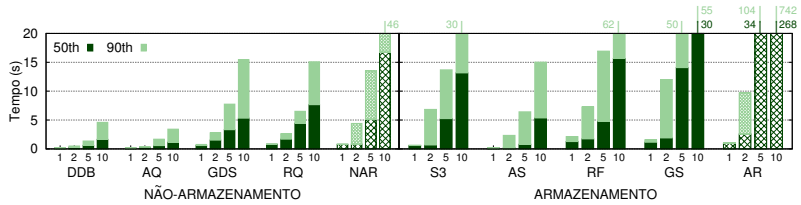


Fig. 3. Latência da obtenção do trinco na presença de concorrência de até 10 clientes.

que o RQ. Para o trinco composto usando só serviços de armazenamento em cloud (AR), observámos uma latência 100% pior que no trinco composto NAR. A configuração TR usa os objetos base mais rápidos, apresentando um latência 100% inferior quando comparada com a NAR. Contudo, esta configuração tem uma limitação importante que se prende o facto de usar dois objetos base na mesma cloud (AQ e AS), o que conduz a uma menor diversidade. Pode verificar-se ainda que o algoritmo do DepSky (TD) tem resultados de aproximadamente dobro da latência da configuração AR (com o o seu comportamento é comparável), e o quadruplo da NAR. Isto acontece porque o algoritmo do DepSky acede ás clouds por fases, enquanto que os restantes (AR, NAR e TR) executam os algoritmos de trinco em paralelo. Por fim, verificámos que o algoritmo de exclusão mútua do SCFS (TS) é o que apresenta melhores resultados. Contudo, se o nosso protocolo oferece uma muito melhor relação custo/desempenho se tivermos em conta todos os custos associados à manutenção dos servidores do SCFS.

Como também se pode ver na figura, o padrão observado na obtenção do trinco verifica-se também nas restantes operações. A única excepção aparece no comportamento do TD quando comparado com as configurações NAR e AR. Isto justifica-se com o facto de todos os objetos base, além de libertarem o trinco, efectuarem também a remoção de entradas de trinco inválidas.

5.2 Execuções na presença de concorrência

Um aspecto importante a ter em conta nos algoritmos de exclusão mútua é a sua capacidade da solução escalar quanto ao número de clientes a tentar obter o trinco concorrentemente. Para avaliar a nossa solução neste sentido, realizámos um teste em que vários clientes (1, 2, 5 e 10) tentam obter o trinco (e em caso de sucesso, o libertam no instante seguinte) e medimos tempo necessário para cada obtenção bem sucedida. No caso dos algoritmos que garantem apenas *liberdade de obstrução* usámos um tempo de *backoff* aleatório de entre 0 e 1 segundos.

A Figura 3 mostra os resultados obtidos os objetos base e para várias configurações de objetos compostos. Novamente, os serviços que não são de armazenamento (à esquerda) fornecem melhores resultados do que os serviços de armazenamento em cloud. Isto acontece porque os serviços do primeiro grupo satisfazem a propriedade *deadlock-freedom*, garantindo que se um conjunto de processos tentar obter um trinco concorrentemente, algum deles irá ter sucesso. Esta propriedade torna-os muito melhores para lidar com concorrência quando comparados com os restantes (ver Tabela 1). O objecto composto (NAR e AR) também garante

apenas *liberdade de obstrução*, tendo assim um aumento linear da latência na obtenção de trincos quando na presença de concorrência. Embora este algoritmo seja rápido com 1 ou 2 clientes, é visivelmente lento com 5 e 10 clientes.

6 Trabalhos Relacionados

Um dos pontos a ter em conta no desenho de serviços de armazenamento é forma como pretende coordenar acessos concorrentes aos seus recursos. Alguns dos trabalhos que propõem este tipo de algoritmos para implementar sistemas seguros são [8,11,15,20]. Em alto nível, o algoritmo de exclusão mútua apresentado neste artigo é muito similar ao apresentado em [20], que por sua vez se assemelha a algoritmos clássicos (desde os anos 80) de exclusão mútua baseada em quoruns. Contudo, este algoritmo requer servidores que executam código para resolver parte do algoritmo [20]. Em contraste, o nosso é construído em cima de algoritmos desenhados especificamente para os serviços de cloud disponíveis nos dias de hoje.

Como dito anteriormente, outro trabalho que motivou o desenvolvimento do nosso algoritmo foi [8]. Este é um protocolo bizantino de consenso baseado em discos partilhados não confiáveis (tais como as clouds de armazenamento). Uma das suas características principais é o facto de poder ser usado para implementar exclusão mútua satisfazendo a propriedade *liberdade de impasse* (que é mais forte que a *liberdade de obstrução*). No entanto, este necessita de um número elevado de acessos às clouds: pelo menos 5 acessos por ronda (podem haver várias rondas se existir concorrência). Por outro lado, o nosso protocolo composto requer apenas entre 2 a 4 acessos por objecto base para obter o trinco (ver tabela 1).

Segundo sabemos, existem apenas dois algoritmos de trinco centrados em dados tolerantes a faltas [11,15], sendo que ambos são baseados no uso de clouds de armazenamento ou de discos. O algoritmo de trinco apresentado por [15] tem duas diferenças importantes quando comparado com o algoritmo de trinco composto apresentado neste artigo. Em primeiro lugar, não garante um trinco sempre seguro no sentido em que o algoritmo admite a possibilidade de existência de mais do que processo com trinco válido. Isto acontece porque a única garantia fornecida por [15] é que, algures no tempo, apenas um processo irá ter o trinco. Este tipo de garantia é adequada para implementar eleição de líder, no entanto é inadequado para garantir exclusão mútua. Em segundo, ele apenas tolera falhas por paragem (não falhas bizantinas). Por outro lado, o algoritmo de exclusão mútua do DepSky [11] tolera faltas bizantinas, sendo o mais similar ao nosso do que diz respeito às garantias fornecidas. Todavia, o nosso algoritmo tem um desempenho quatro vezes superior e não necessita que os clientes tenham os relógios sincronizados.

7 Conclusões

Existem vários serviços que necessitam coordenar o acesso concorrente de clientes a recursos partilhados (e.g.,ficheiros). No entanto as soluções existentes são ou ineficientes ou têm elevados custos associados. Este problema levou-nos a desenhar um novo protocolo de exclusão mútua tolerante a faltas bizantinas que não necessita de nenhum servidor específico a correr nas clouds. A avaliação mostra que

este protocolo tem um desempenho, por um lado muito superior ao dos protocolos existentes com comportamento semelhante, por outro comparável a soluções que mantém servidores dedicados a correr nas clouds.

Agradecimentos. Queremos agradecer ao Vinícius Cogo pela sua ajuda em vários aspectos do artigo. Este trabalho foi suportado pela Comissão Europeia através dos projectos BiobankCloud (FP7/ICT-317871) e SuperCloud (H2020/ICT-643964), e pela Fundação para a Ciência e a Tecnologia (FCT) através de seu programa multianual (LaSIGE).

Referências

1. Amazon DynamoDB. `http://aws.amazon.com/dynamodb/`.
2. Amazon S3. `http://aws.amazon.com/s3/`.
3. Google Cloud Datastore. `https://cloud.google.com/datastore/`.
4. Google storage. `https://developers.google.com/storage/`.
5. Message queuing service with simple API – Rackspace cloud queues. `http://www.rackspace.com/cloud/queues/`.
6. Microsoft Azure Queue. `http://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-queues/`.
7. Rackspace cloud files. `http://www.rackspace.co.uk/cloud/files`.
8. I. Abraham, G. Chockler, I. Keidar, and D. Malkhi. Byzantine disk paxos: optimal resilience with byzantine shared memory. *Distributed Computing*, 18(5):387–408, 2006.
9. H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon. RACS: A case for cloud storage diversity. In *Procs. of the ACM SoCC*, 2010.
10. H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. Wiley Series on Parallel and Distributed Computing. Wiley-Interscience, 2nd edition, 2004.
11. A. Bessani, M. Correia, B. Quaresma, F. Andre, and P. Sousa. DepSky: Dependable and secure storage in cloud-of-clouds. *ACM Transactions on Storage*, 9(4), 2013.
12. A. Bessani, R. Mendes, T. Oliveira, N. Neves, M. Correia, M. Pasin, and P. Verissimo. SCFS: a shared cloud-backed file system. In *Procs. of the USENIX ATC*, 2014.
13. A. Bessani et al. The overbank cloud architecture, protocols and middleware, Nov. 2014. Deliverable D4.2 of BiobankCloud project.
14. B. Calder et al. Windows Azure storage: a highly available cloud storage service with strong consistency. In *Procs. of the ACM SOSP*, 2011.
15. G. Chockler and D. Malkhi. Light-weight leases for storage-centric coordination. *International Journal of Parallel Programming*, 34(2), Apr. 2006.
16. J. Y. Chung, C. Joe-Wong, S. Ha, J. W.-K. Hong, and M. Chiang. Cyrus: Towards client-defined cloud storage. In *Procs. of the ACM EuroSys*, 2015.
17. C. Gray and D. Cheriton. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proc. of the ACM SOSP*, 1989.
18. M. Herlihy. Wait-free synchronization. *ACM TOPLAS*, 13(1):124–149, 1991.
19. P. Hunt, M. Konar, F. Junqueira, and B. Reed. Zookeeper: Wait-free coordination for internet-scale services. In *Procs. of the USENIX ATC*, 2010.
20. D. Malkhi and M. Reiter. Secure and scalable replication in Phalanx. In *Procs. of the IEEE SRDS*, 1998.
21. T. Oliveira, R. Mendes, and A. Bessani. Sharing files using cloud storage services. In *DIHC, co-located with Euro-Par*, 2014.

Asynchronous Parallel Ant-Colony Optimization Strategies: Application to the Multi-Depot Vehicle Scheduling Problem with Line Exchanges

David Semedo, Pedro Barahona, and Pedro Medeiros

NOVA-LINCS DI-FCT-UNL
Laboratory for Computer Science and Informatics
Faculdade de Ciências e Tecnologia, 2829-516 Caparica, Portugal
df.semedo@campus.fct.unl.pt
{pb,pdm}@fct.unl.pt

Abstract. The Multi-Depot Vehicle Scheduling Problem with Line Exchanges is an NP-*hard* combinatorial optimization problem arising in transit service companies.
Due to the complexity of the problem and the large size of the search space, complete search cannot be used in order to guarantee optimality. Hence, we adopted the Ant-Colony Optimization (ACO) metaheuristic. ACO can be improved using parallel computing. However, in order to maintain the semantics of the original ACO metaheuristic, parallelization strategies need to be synchronous, originating the straggler problem. We propose three shared-memory asynchronous parallelization strategies that break the original ACO algorithm semantics and differ essentially on the degree of concurrency allowed while manipulating the learned information.
The results show that two of our asynchronous strategies outperform synchronous ones in terms of the speedup achieved and also solution quality. Additionally, the speedup increases as the level of concurrency allowed in asynchronous strategies increases, without sacrificing the quality of the solutions obtained. The proposed strategies effectiveness is validated through an analysis and discussion of the ACO convergence and scalability, in terms of search depth and number of threads used in a *shared-memory* architecture, with *multi-core* processors.

Keywords: Multi-Depot Vehicle Scheduling Problem (MD-VSP), Parallel Computing, Asynchronism, Concurrency, Ant-Colony Optimization (ACO)

1 Introduction

The Multi-Depot Vehicle Scheduling Problem with Line Exchanges (MD-VSPLE), a variant of the more general Vehicle Scheduling Problem (VSP), is an NP-*hard* [1, 12] combinatorial optimization arising in transit services companies, which consists of producing a schedule for each vehicle such that each trip is

covered by only one vehicle and the operational costs are minimized, whilst satisfying a set of constraints. In this variant, more than one depot and line exchanges are considered. Additionally, we assume that the fleet of vehicles is heterogeneous (i.e. there are several types of vehicles each with different characteristics and constraints).

In a real life context, instances of this problem, specially in urban areas, have a large number of variables, which, due to the problem complexity and the additional constraints taken into account in this work, makes impractical solving it to optimality.

Metaheuristics [10] are defined as methods that orchestrate an interaction between local improvement procedures (e.g. Local Search algorithms) and higher level strategies in order to develop search strategies capable of escaping local optima and guiding the search towards promising regions of the search space, by sacrificing optimality. The Ant-Colony Optimization (ACO) metaheuristic, originally proposed by Dorigo [9], is a metaheuristic inspired in the behaviour of real ants which use pheromones as a communication medium.

A stochastic learning procedure allows the agents to build a wide variety of solutions, leading to a better (in depth) exploration of the search space, than greedy procedures. Agents search experience influences future iterations which makes ACO similar to reinforcement learning procedures.

Despite of the proven capabilities of ACO [14], for complex problems with large search spaces, as the one we address in this work, it may fail to find quality solutions in a reasonable amount of time.

Parallel computing is commonly used to improve ACO effectiveness in terms of computation time, solution quality or both. Parallel strategies [15, 6] can target either *shared-memory* (SM) or *distributed* architectures, either using CPUs, GPUs or both. Distributed architectures offer better scalability than SM ones, however, communication within nodes is more expensive due to the necessity of transferring data between disjoint address spaces. ACO agents rely on a high-level of communication in order to report their findings and improve the learned information about the search space. Therefore, in this paper we target SM architectures.

Despite of the architecture used, in order to preserve the semantics of the original ACO metaheuristics, parallel strategies must be synchronous, i.e., a synchronization point must exist after each iteration where the new information obtained by each agent must be used to update the global learned information. We propose asynchronous strategies, that break the semantics of the original metaheuristic in order to achieve better performance, and differ on the degree of concurrency allowed while manipulating the learned information. Since synchronization points after each iteration are removed, it is possible to achieve more parallelism. We propose a set of modifications to the original procedure in order to preserve convergence properties.

The proposed strategies effectiveness is validated through an analysis and discussion of the ACO convergence and scalability, in terms of search depth and

number of threads used, in a *multi*-processor SM architecture, with *multi-core* processors. We compare the results achieved with a synchronous ACO version.

We also analyse the *single*-processor architecture with *multi-core* processors due to the fact that it corresponds to common personal computers (PC) architectures. For companies that face the MD-VSPLE problem, it would be of great value to have a decision support tool capable of effectively solving this problem and which would not require expensive hardware.

The remainder of this paper is structured as follows. Section 2 describes and formalizes the MD-VSPLE as well as additional constraints taken into account. The sequential algorithm using ACO will also be described in this section. Section 3 presents a review of the literature on parallel ACO. In section 4 we present our proposed asynchronous strategies and we discuss important implementation details. In section 5 experimental results are presented and discussed. Finally, section 6 concludes the paper giving an outlook on future research.

2 Multi-Depot Vehicle Scheduling Problem with Line Exchanges

The MD-VSPLE is an NP-*hard* [1, 12] problem in which the objective is to find a schedule that minimizes the operational costs, whilst satisfying a set of constraints. Since it is our goal that the variant addressed in this work matches real life situations, we also take into account an heterogeneous fleet, in which for each type of vehicle, additional constraints are enforced. To the best of our knowledge, the variant of the MD-VSPLE addressed in this work has not been addressed yet.

2.1 Problem Formulation

The MD-VSPLE can be formulated as follows:

Let L be a set of lines to be covered and T be a set of n time-tabled departures $T_1, \ldots, T_n$, such that for each $j \in [1, n]$, s_j and e_j denote the starting and ending time, respectively, it_j and ft_j denote the initial and final terminus, respectively, and $l_j \in L$ the corresponding line of departure j. Let D be a set of m depots $D_1, \ldots, D_m$ where each depot D_k has a total capacity of r_k (with $r_k \leq n$) vehicles.

Let $\tau_{i,j}$ be the travel time for a vehicle to go from the arrival terminal of departure T_i to the terminal of departure T_j, where $i, j \in [1, n]$. A pair of trips (T_i, T_j) is said to be *compatible* iff the same vehicle can cover trips T_i and T_j in the sequence, i.e, the following inequality is verified:

$$e_i + \epsilon + \tau_{i,j} \leq s_j \tag{1}$$

where ϵ is the minimum time between an arrival and a departure, in order to allow preparation of the vehicle and/or driver exchange. A vehicle *scheduling* S_l, performed by a vehicle v_i stationed at depot D_k, consists of a sequence of tasks.

A task may correspond to performing a departure (*service*), stay idle, a trip without passengers to/from a depot, a line exchange or a maintenance episode. Departures tasks in S_l must be pair-wise compatible departures, assuming the S_l sequence order.

The following hard-constraints are enforced: each departure is covered by exactly one vehicle, each vehicle is assigned to a *schedule* with pairwise compatible departures, returning to its depot at the end of the *service*.

We also take into account additional constraints like the maximum daily working time for each vehicle type, the obligation and duration of maintenance episodes and the amount of time a vehicle can stay idle in a terminal.

Objective Function. The objective function F to be minimized takes into account the investment cost of each vehicle measured by its daily depreciation (€/day) (including the cost of maintenance episodes) and the running cost of the total distance travelled by each vehicle. Each of the costs referred change according to the vehicle type.

Let Nb be the total number of vehicles used (i.e. the total number of services in the solution). The function F of a solution *sol* is defined as follows:

$$F(sol) = \sum_{i=1}^{Nb} distTravelled(v_i) * distCatCost(v_i) + dailyCatCost(v_i) \qquad (2)$$

where $distTravelled(v_i)$ is the total distance travelled by the vehicle v_i in the solution *sol*, $distCatCost(v_i)$ and $dailyCatCost(v_i)$ is the running cost and the daily depreciation, respectively, of vehicles from the category of vehicle v_i. The unused vehicles slots in each depot D_k do not contribute to the total cost.

2.2 Ant-Colony Optimization Metaheuristic

The ACO metaheuristic is a population-based metaheuristic for solving hard-combinatorial optimization problems based on a colony of agents that construct solutions using a pheromone model that corresponds to a parametrized probability distribution over the solution space, adding components to a partial solution. These components are chosen based on heuristic information of the problem and pheromone trails.

Pheromone trails are modelled as distributed numerical information of the search space graph, which varies according to the problem being solved. Agents cooperatively update pheromone trails at runtime in order to reflect their search experience. The stochastic component is essential to achieve solution diversity and avoid being stuck at a local optima.

The first ACO implementation proposed was the Ant System (AS) algorithm [8] which was applied to the Travelling Salesman Problem (TSP). Based on the AS, more sophisticated implementations emerged, such as the MAX-MIN ACO [16] and the Ant Colony System [7] (ACS) algorithms.

The base algorithm scheme, shown in algorithm 1, is roughly the same.

Algorithm 1 Ant-Colony Optimization.[10]

```
1: procedure ANT-COLONY OPTIMIZATION(problem) return best solution found
2:     Initialization
3:     while termination condition not met do
4:         ConstructAntSolutions
5:         UpdatePheromoneTrails
6:     end while
       return best solution
7: end procedure
```

In the *Initialization* step pheromone variables are initialized. Then, it iterates until some termination criteria is met (e.g. stop after P iterations). In the *ConstructAntSolutions* a set of N agents perform a constructive search procedure guided by heuristic information and by the pheromone trails. Lastly, in the *UpdatePheromoneTrails* step, the pheromone trails are updated such that good solutions will be more desirable. In order to avoid a fast convergence of all the agents towards sub-optimal solutions, pheromone evaporation is applied.

2.3 Sequential ACO Algorithm for the MD-VSPLE

By sacrificing optimality, efficient approximative algorithms, capable of achieving high-quality solutions in a reasonable amount of time can be developed. Approximative algorithms can be either *constructive*, in which solutions are built incrementally, or *Reparative*, in which the algorithm attempts to improve a complete solution at each iteration.

Reparative procedures require the definition of a move operator which is used to perform local moves from one solution to another, at each iteration. Due to the complexity of our problem, more concretely to the large number of constraints involved, it is not trivial to design a move operator that keeps the solutions consistent and without constraint violations, also taking into account that the operational costs must be minimized. Therefore, *constructive* approaches should be more adequate and yield better results in our problem.

We thus adopt the ACO since it not only uses a *constructive* approach but also uses a reinforcement learning procedure that makes the ACO a superior algorithm when compared to greedy ones.

ACO for MD-VSPLE. Our algorithm implements the AS algorithm, which despite of the fact that it may be not as robust as the MAX-MIN and ACS algorithms in its sequential versio (as discussed in section 4) it is more suitable for parallel implementations.

The pheromone trails model should hold information of the desirability of each solution component. Each component added to a solution corresponds to an assignment of a vehicle to a departure. Let P be the set of possible locations from where a vehicle can be picked when assigning a vehicle to a given departure. Our model corresponds to a $|P| \times |T|$ matrix, i.e., we assign a pheromone value

τ_{ij} to each pair $\langle l_i, T_j \rangle$ that denotes the desirability of assigning vehicles that are in a location $l_i \in P$ to a departure $T_j \in T$.

At each step t each agent k uses a probability distribution to select the solution component c_i^j to be added to a partial solution s_p, i.e, from which location $l_i \in P$ should a vehicle be picked to a departure T_j. We use the most widely used probability distribution, from AS. Let $Admiss_k$ be the set of vehicles that can be assigned (do not violate any hard constraint) to a departure T_j. The probability distribution is defined as follows:

$$p_k(c_i^j|s_p) = \begin{cases} \dfrac{\tau_{ij}^{\alpha} \cdot \left[\eta(c_i^j, v_a)\right]^{\beta}}{\sum_{l \in Admiss_k} \tau_{lj}^{\alpha} \cdot \left[\eta(c_l^j, v_a)\right]^{\beta}} & , \ v_a \in Admissible_k \\ 0 & , \ otherwise \end{cases} \tag{3}$$

where $\eta(\cdot)$ is a function that assigns to each c_i^j in which a vehicle $v_a \in Admiss_k$, an heuristic value that evaluates the quality of this assignment in the context of the problem, i.e., corresponds to the utility value of a given vehicle to perform a given departure. The α and β parameters are used to control the influence of pheromone values and heuristic information, respectively, on the algorithm behaviour.

The utility $\eta(c_i^j, v_a)$ of a vehicle v_a located on a terminal tm_o, performing a departure j starting on terminal tm_j, is a function $\eta : \mathbb{R}_{\geq 0} \mapsto]0, 1]$ defined as follows:

$$\eta(c_i^j, v_a) \ = \ e^{\dfrac{-dist(tm_o, tm_j) * distCatCost(v_a) * k_1 + dTerm(tm_o, tm_j) * k_2}{k_3}} \tag{4}$$

where $dist(t_1, t_2)$ is the distance from t_1 to t_2, $dTerm(t_1, t_2)$ is a function that returns 1 if $t_1 = t_2$ or 0 otherwise, and k_w, where $w \in [1, 2, 3]$, are weights that can be parametrized.

In the end of an ACO iteration pheromone trails are evaporated and updated. Pheromone evaporation is implemented as follows:

$$\tau_{ij} \leftarrow (1 - \rho) * \tau_{ij} \tag{5}$$

where $0 < \rho \leq 1$ is a parameter that denotes the rate of evaporation. If a solution component is not used by an agent, its corresponding pheromone value τ_{ij} decreases exponentially in the number of iterations, allowing the algorithm to avoid bad solution components, since its desirability is reduced.

After performing the pheromone evaporation step, the pheromone values are updated with learned information from each agent. The update is defined as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1} \Delta\tau_{ij}^k \tag{6}$$

where $\Delta\tau_{ij}^k$ denotes the amount of pheromone deposited by agent k on the solution components that belong the solution s_k obtained:

$$\Delta\tau_{ij}^k = \begin{cases} \dfrac{\lambda}{F(s_k)} & , \ c_i^j \in s_k \\ 0 & , \ otherwise \end{cases} \tag{7}$$

where λ is a parameter used to control the amount of pheromone deposited. With this expression, the amount of pheromone deposited by each agent depends on the quality of the solutions obtained. Therefore, agents that obtain better solutions will deposit a larger amount of pheromone, favouring good solution components.

ACO Agent algorithm. As previously stated, each agent k constructs a solution incrementally by adding solution components iteratively and using the probability distribution p_k. Departures are sorted by starting minute. At each iteration of the agent algorithm, a set of admissible vehicles is built. Admissible vehicles are vehicles that do not violate any hard-constraint if assigned to the departure being addressed. While building the set of admissible vehicles, the algorithm sends to maintenance vehicles that need maintenance, i.e., vehicles that would violate the maintenance constraint if they performed the departure.

In this paper we explore parallel ACO metaheuristics, in order to improve the presented ACO algorithm performance. Concretely, we propose a new parallel ACO algorithm that explores the fact that the pheromone update and evaporation mechanisms demand a synchronization point at the end of an iteration, originating the straggler problem.

3 Related Work

Parallel computing has been used as a mean of improving ACO algorithms performance, therefore, several different algorithms using different strategies have been proposed [15]. The most promising models for achieving high computational efficiency are the *coarse-grain master-slave*, in which a master process that manages global information controls a group of search processes, and *multicolony*, in which each processor executes a colony.

In [5] the authors propose an SM ACO algorithm that uses the *coarse grain master-slave* model and compare SM architectures with distributed architectures. The SM version of the proposed algorithm outperformed the distributed version. The main reason is the communication overhead imposed by the network.

Delisle et al. [4] presented an SM parallel implementation using OpenMP of ACO in order to solve an industrial scheduling problem. As the authors stated, the ACO algorithm follows exactly a *fork-join* scheme. Additionally, despite the fact that with SM no explicit communications are needed, therefore omitting communications overheads, a synchronization barrier that occurs when each ant

finishes the search and the pheromone matrix needs to be updated, causes the straggler problem and prevents the algorithm from achieving better performance.

Cipar et al. [3] defined the straggler problem as the situation in which a small number of threads (the stragglers) take longer than the others to execute a given iteration. Since all the threads will eventually be synchronized, all threads will proceed at the speed of the slowest one. As stated in [2] one of the solutions for the straggler problem is to make the algorithm asynchronous. Despite of potentially making the algorithm more complex in order to maintain its semantics and properties, it eliminates completely the straggler problem, and allows the algorithms to use all the available parallelism.

In the context of ACO and in a distributed architecture, Kotsis et al.[11] proposed a *partially asynchronous* parallelization scheme on the AS algorithm to reduce the straggler problem caused by the ACO sychronization barrier. The scheme consists in creating temporary sub-colonies in computer nodes which live for a certain number of iterations. Despite reducing the amount of communication performed the straggler problem still occurs within each sub-colony.

In [17] the authors propose two ACO asynchronous models based on the *cunning* Ant System (cAS) algorithm. The difference in the two models consists in modelling the pheromone matrix as a critical section (AP-cAS) or not (RAP-cAS). The RAP-cAS model achieved the most promising results (in terms of speedup) comparing with (AP-cAS) and a synchronous model. In order to use the cAS algorithm, the problem model must target a *reparative* search scheme, which as discussed previously in section 2.3, is not suited for our problem. Additionally, asynchronism breaks the semantics of the original ACO algorithm and the convergence of the proposed algorithms was not analysed.

4 Parallel Ant-Colony Metaheuristic Models

The *MAX-MIN* and the *ACS* algorithms include mechanisms to avoid premature convergence and consequently may achieve better results than AS, in a sequential setup. However, these mechanisms affect negatively the exploitation of parallel resources: in *MAX-MIN* and *ACS* only the best solution obtained in an iteration is used to perform the update, what forces the master to always wait for all the solutions. Additionally in *ACS* search agents perform local updates while building solutions, originating concurrency on the pheromone matrix. Therefore, AS is more suitable to design an asynchronous parallel strategy. Our algorithms follow the *coarse-grain master-slave* model.

4.1 Asynchronous Model

The main modification of our asynchronous model is the following: the master generates a set of $P \times N$ tasks, where P is the total number of iterations. Each task belongs to a given iteration $p \in P$ and a given search agent $n \in N$. These tasks are executed asynchronously and the results are sent to the master.

It may now occur that a search agent that is constructing a solution views an update to the matrix. This is the situation that breaks the original semantics of the ACO algorithm and can potentially affect negatively the convergence. We show in the next section that this is not the case.

The pheromone update and evaporation steps must be reviewed in order to allow ACO to converge. The following modifications were made:

Pheromone Update - When an agent finishes the search, the master is notified and performs a partial update based on the generated solution;

Pheromone Evaporation - In the original AS algorithm the total number of evaporations performed is equal to P. Since now the agents execute asynchronously it is not trivial to design an equivalent scheme, since agents from an iteration p_1 with $p_1 > p_2$ may finish first than agents from an iteration p_2 and we do not know in each iteration how many agents will finish first, since it depends on the real-time decisions made by the operating system scheduler. In order to achieve an approximate behaviour, an evaporation step is performed after $N/2$ search agents from an iteration p, finish their execution. This assumes that some agents will be slower. With this strategy, evaporation steps may be performed sooner than they would be on the original algorithm, however this is not a problem, since there is an attempt to perform evaporation steps after an average of x agents have finished, although this is not guaranteed. Ideally an evaporation step would be performed every time x search agents finish. We believe that our proposed scheme is a good approximation of this behaviour and its contribution to the algorithm convergence is evaluated in section 5.

Despite of the new algorithm conceptual simplicity, it introduces concurrency on the pheromone matrix. The master will be regularly performing updates while agents will be reading the matrix contents.

Concurrency Models. We propose three different concurrency control models, that differ on the degree of concurrency allowed while manipulating the pheromone matrix. Each model affects the original algorithm semantics in a different manner, therefore, the algorithm convergence for each model must be assessed.

Blocking Matrix (AsyncBM) - In this model when a master is performing updates the whole matrix is blocked and none of the agents can proceed until the update is performed. With this model if an update is performed, all the agents will receive the complete update of all the solutions components;

Blocking Column (AsyncBC)- When an agent is choosing a component c_i^j to a partial solution, it will read an entire column of the pheromone matrix, i.e., selecting from which location i should a vehicle be picked to perform the departure j. In this model, instead of blocking the entire matrix, only the column that is being updated/read is blocked. This ensures that if updates/reads are going to be concurrently performed, each agent will decide

on each assignment based either on the new or the old information, but not on a mix of both;

Lock-Free (AsyncLF)- In the limit, we could allow all the concurrent updates and a search agent can possibly use both information before an update and information after an update, on the same decision. Conceptually this achieves better diversification and in practice maximizes concurrency. This model assumes that reads and writes operations are atomic.

4.2 Technical and Implementation issues

The synchronous ACO algorithm was implemented using OpenMP. The implementation is straightforward. It simply consists in adding a *pragma* annotation in the loop where search agents are launched (line 4 of Algorithm 1).

The base asynchronous algorithm design is based on the thread pool, avoiding consecutive thread creation and destruction overhead, in which the results are placed on a pool of solutions. The overall scheme corresponds to a *1-producer/N-consumers* pattern.

The tasks are placed on a pool of tasks implemented with a lock-free queue, from which threads in the thread pool will consume tasks. When a thread finishes processing a task, i.e., builds a solution, the solution is placed on a solutions pool and the master consumes it by performing the pheromone update and evaporation steps.

In order to achieve high performance, the implementation must be efficient. The proposed concurrency models are implemented as follows:

Blocking Matrix - Each time the master or one of the threads want to access the pheromone matrix they have to acquire a lock. This lock is implemented as a *spin lock* [13];

Blocking Column - In this model we have an array of *read-write* locks with T elements, where each element is the lock of a column;

Lock-Free - The lock-free model is implemented by using C++ *atomics*, ensuring that reads and writes operations are atomic.

5 Results

In this section we present and discuss the results obtained from the tests performed. We implemented a sequential version of the AS algorithm (ASSeq). This sequential implementation was parallelized using OpenMP (ASSync). Our proposed algorithms were implemented in C++.

The main tests were performed on a 4-node NUMA machine with 4 AMD(R) Opteron 6272 processors @ 2.1 GHz, each with 16 cores, and with 64GB RAM (16GB for each NUMA node). We also analysed the execution times on a PC with an Intel(R) Core i7-4700MQ CPU @ 2.40GHz supporting Hyperthreading(R) and with 32GB RAM. The instance of the MD-VSPLE addressed corresponds to a subset of 6 CARRIS[1]bus lines with ascending and descending directions,

with a total of 732 departures and 12 terminal locations. Additionally, 3 vehicle categories and 2 depots are considered, each with 100 vehicles of each category.

Due to the stochastic nature of the algorithms and to achieve more robust results, 5 runs are performed for each configuration.

5.1 Convergence Analysis

In order to assess our algorithm convergence we performed a test in which the number of iterations P is 250 and the number of search agents K is 16. The number of threads used is irrelevant since we are not analysing the speedup. For each iteration $p \in [1, 250]$ of the search, we computed the mean of the objective function of all the 16×5 generated solutions from the 16 agents and 5 runs. We plotted the mean values for each p iteration and we applied a linear regression on the data. The results can be seen in Figure 1. In this figure four plots are shown, one for the ASSync and one for each of our concurrency models. The slopes of the linear regression lines (blue lines) are shown and can be interpreted as a measure of the *intensity* of convergence. The red line shows the best solution obtained during each iteration p. The best solution objective value achieved is shown in the plot title.

We observe that our proposed asynchronous strategies are able to converge as the number of iterations increases. It is worth noting that both the AsyncLF and AsyncBC algorithms outperformed ASSync in terms of convergence intensity and solution quality. The AsyncBM algorithm was not superior in terms of solution quality but was very close to ASSync. The AsyncLF algorithm not only presents the highest convergence (slope of ~ -0.22) but also achieved the best solution in the first 100 iterations. However, it takes longer to converge than the other algorithms since only after iteration 150 the algorithm starts achieving several quality solutions. The AsyncBC and AsyncLF algorithms start achieving quality solutions in less iterations and should be preferred if one wants a good solution quickly.

The ability of achieving good diversity while generating solutions is crucial since it corresponds directly to a better exploration of the search space. As the degree of concurrency increases, search agents that are executing receive updates from agents that have already finished sooner and, this allows the executing agents to perform *better* decisions sooner, i.e., agents start exploring promising regions sooner which increases the chances of achieving quality solutions. Furthermore, since synchronization barriers where each agent receives a copy of an updated matrix are removed and search agents from the same iteration may use different numerical values for the same decision, better diversity is achieved.

5.2 Performance Analysis

In our performance experiments we target the following algorithms: ASSeq, ASSync and our three proposed parallel asynchronous algorithms.

[1] A portuguese bus service transport company.

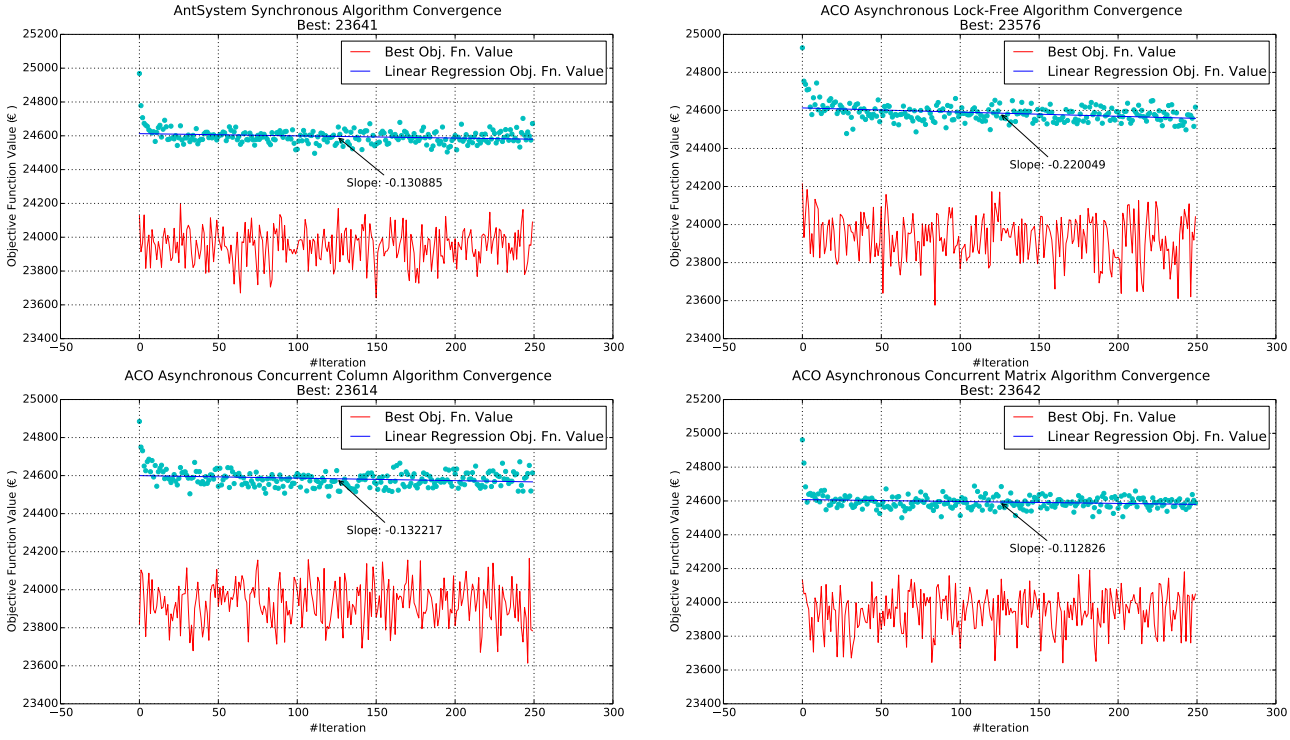


Fig. 1: Convergence Analysis of proposed algorithms with 250 iterations.

The first experiment was performed on the 64 core machine using 64 threads. With this experiment we intend to analyze how the speedups are affected when the search goes deeper (in number of iterations). We run each algorithm using $p \in [10, 50, 100, 250, 500]$ iterations. This experiment was performed with $4, 8, 16$ and 32 search agents. Figure 2 shows the result of the experiment. We observe that in all cases both AsyncLF and ASyncBC algorithms achieves better speedups than ASSync. The AsyncBM algorithm proved to be unstable yielding better speedups than ASSync with 8 and 16 agents but not with 32 and 64.

With 8, 16 and 32 search agents there are threads that were launched but are not working since the number of threads is greater than the number of tasks to be executed. With 64 agents this situation does not occur and each thread gets atleast one agent from each iteration. This explains why speedups are slightly better in general with 64 agents (except for AsyncBM). We observe that AsyncLF achieves almost always better speedups than the other asynchronous strategies. As we expected, and due to the fact that the whole matrix is blocked, the AsyncBM algorithm is the worse asynchronous algorithm. In general and as we expected, we verify that as more concurrency is allowed, the better are the speedups achieved.

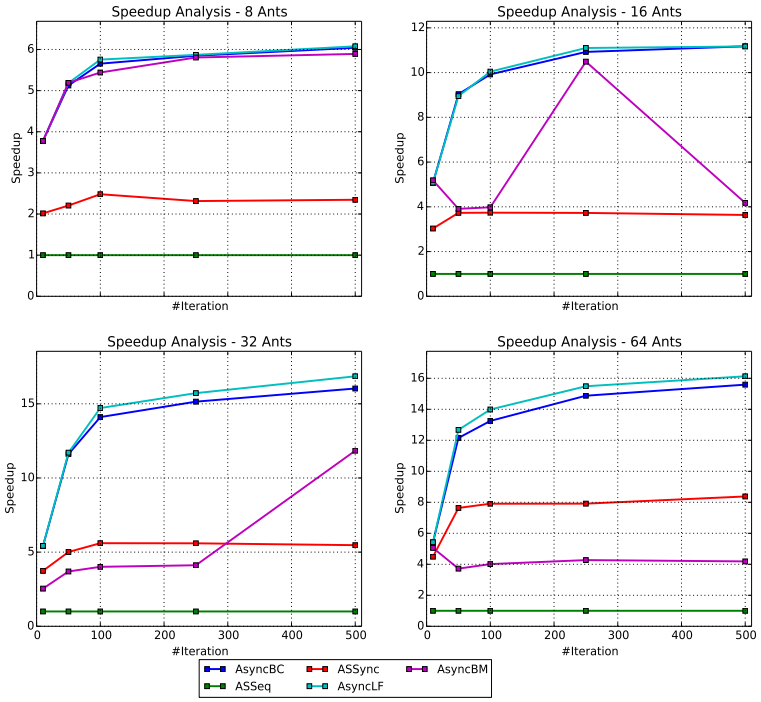


Fig. 2: Speedup Results with 64 threads.

For the second experiment, also performed on the 64 core machine, the number of iterations was fixed to 1000. This experiment aims to evaluate the scalability of our algorithms as the number of threads increases. We run each algorithm using $th \in [2, 4, 8, 16, 32, 64]$ threads and, for each number of threads th we measured the speedup achieved. Like in the previous experiment, we tested with $4, 8, 16$ and 32 search agents. Figure 3 shows the result of the experiment.

We observe a speedup increase as the number of threads increases. In all configurations, the overhead of concurrency mechanisms is only observed when more than 8 threads are used. Like in the previous experiment, the best results were achieved with 64 search agents per iteration. Both results indicate that as we increase the amount of work in each iteration, better speedups are achieved, i.e., parallel resources exploitation is more effective.

Despite of the overhead introduced by read/write locks, the AsyncBC performance is similar to the AsyncLF algorithm, with the last being slightly better. We observe that both AsyncBC and AsyncLF algorithms are able to scale in terms of workload and number of threads. Furthermore, both algorithms scale almost logarithmically with 32 and 64 agents as the number of threads increases, suggesting that better results would be achieved on a machine with more cores and using more threads.

The AsyncBM algorithm is always worse than the other asynchronous models and sometimes even worse than SyncAS, revealing that blocking the entire

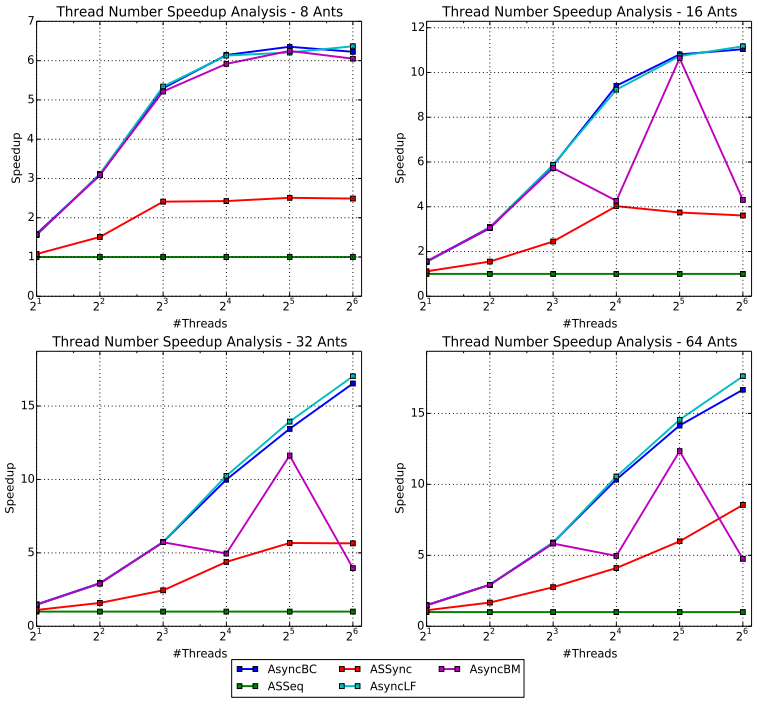


Fig. 3: Speedup Results with 1000 iterations.

matrix has a significant negative impact on the performance. The maximum speedup achieved was $\sim$ 17.6x with 64 threads and 64 agents on a 64 core machine, what gives an efficiency of $\sim$ 0.364. Even for the best configuration (64 agents) efficiency decreases as the number of threads increases. A strict comparison with other proposed algorithms is not possible to stablish since no ACO parallel algorithm was developed and applied to the MD-VSPLE.

It is worth noting that in our first implementation we were only able to achieve speedups smaller than 5.5x with CPU cores usage being very low. We performed some experiments and we found out that concurrent memory allocations have a great impact on the performance (*malloc*(3) contention). In our implementations we attempted to minimize the number of memory allocations on parallel regions, however, the remaining allocations still affect the performance. Still, the improvements were significant.

Our last experiment intends to evaluate the capability of our best algorithm (AsyncLF), using 8 search agents and 8 threads, of performing a moderate/high depth search in a reasonable amount of time on a PC. Table 1 presents the results. We verify that our algorithm is able to achieve more than 4x speedup for moderate/high search depth (500/1000 iterations) on a processor with 4 cores, with Hyperthreading(R), and taking only a few seconds to execute.

Table 1: Results of executing the AsyncLF algorithm on a PC, with 8 search agents and using 8 threads.

#Iterations	Execution Time (s)	Speedup	Best Obj. Function Value (€)
10	2.312	1.47	24154
50	3.542	2.42	23832
100	4.897	3.16	23720
250	9.226	3.91	23653
500	16.456	4.31	23606
1000	30.801	4.55	23540

6 Conclusions and Future Work

In this work we presented a parallel asynchronous ACO algorithm for the MD-VSPLE (taking into account additional real world constraints), based on the AS algorithm and three different concurrency models, mitigating the straggler problem. Additionally, as asynchronism modifies the semantics of the original algorithm, we proposed a set of modifications in order to ensure convergence and effectively explore the search space.

Our experiments show that our proposed asynchronous algorithms not only are able to converge but also achieved better solutions than ASSync. The AsyncBC and AsyncLF algorithms outperform ASSync in terms of speedups achieved and scalability, with the AsyncLF algorithm yielding the best results.

Experiments performed on a PC shown that AsyncLF is able to perform a moderate/high depth search in a reasonable amount of time.

The next step in order to improve our algorithms performance is to use a different *malloc* implementation (e.g. *TCMalloc* or *jemalloc*) instead of the *glib* implementation. Additionally we also intend to perform additional tests in order to identify other aspects that affect negatively the performance. It is also our intention to extend the pheromone trails model and instead of having a row for each location, we want to have the set of all possible vehicles. This model significantly increases the pheromone matrix dimensions and, the computations performed by each search agent become very expensive. We aim to add a second level of parallelism by using GPUs to compute the ACO probability distribution values for each vehicle, at each decision point.

Acknowledgements
This research was partly supported by project "RtP - Restrict to Plan", funded by FEDER (*Fundo Europeu de Desenvolvimento Regional*), through programme COMPETE - POFC (*Operacional Factores de Competitividade*) with reference 34091.

References

[1] Bertossi, A.A., Carraresi, P., Gallo, G.: On some matching problems arising in vehicle scheduling models. Networks 17(3), 271–281 (1987)

[2] Cipar, J., Ho, Q., Kim, J.K., Lee, S., Ganger, G.R., Gibson, G., Keeton, K., Xing, E.: Solving the straggler problem with bounded staleness. In: Proceedings of the 14th USENIX Conference on Hot Topics in Operating Systems. pp. 22–22. HotOS'13, USENIX Association, Berkeley, CA, USA (2013)
[3] Cipar, J., Ho, Q., Kim, J.K., Lee, S., Ganger, G.R., Gibson, G., Keeton, K., Xing, E.P.: Solving the straggler problem with bounded staleness. In: HotOS'13. pp. –1–1 (2013)
[4] Delisle, P., Krajecki, M., Gravel, M., Gagné, C.: Parallel implementation of an ant colony optimization metaheuristic with OpenMP. In: International Conference on Parallel Architectures and Compilation Techniques (2001)
[5] Delisle, P., Gravel, M., Krajecki, M., Gagné, C., Price, W.L.: Comparing parallelization of an aco: Message passing vs. shared memory. In: Proceedings of the Second International Conference on Hybrid Metaheuristics. pp. 1–11. HM'05, Springer-Verlag, Berlin, Heidelberg (2005)
[6] Delévacq, A., Delisle, P., Gravel, M., Krajecki, M.: Parallel ant colony optimization on graphics processing units. Journal of Parallel and Distributed Computing 73(1), 52 – 61 (2013)
[7] Dorigo, M., Gambardella, L.M.: Ant colony system: A cooperative learning approach to the traveling salesman problem. Trans. Evol. Comp 1(1), 53–66 (Apr 1997)
[8] Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: Optimization by a colony of cooperating agents. Trans. Sys. Man Cyber. Part B 26(1), 29–41 (Feb 1996)
[9] Dorigo, M.: Optimization, Learning and Natural Algorithms. Ph.D. thesis, Politecnico di Milano, Italy (1992)
[10] Gendreau, M., Potvin, J.Y.: Handbook of Metaheuristics. Springer Publishing Company, Incorporated, 2nd edn. (2010)
[11] Kotsis, G., Bullnheimer, B., Strauss, C.: Parallelization strategies for the ant system. Technical report (October 1997)
[12] Lenstra, J.K., Kan, A.H.G.R.: Complexity of vehicle routing and scheduling problems. Networks 11(2), 221–227 (1981)
[13] Mellor-Crummey, J.M., Scott, M.L.: Algorithms for scalable synchronization on shared-memory multiprocessors. ACM Trans. Comput. Syst. 9(1), 21–65 (Feb 1991)
[14] Mohan, B.C., Baskaran, R.: A survey: Ant colony optimization based recent research and implementation on several engineering domain. Expert Systems with Applications 39(4), 4618 – 4627 (2012)
[15] Pedemonte, M., Nesmachnow, S., Cancela, H.: A survey on parallel ant colony optimization. Applied Soft Computing 11(8), 5181 – 5197 (2011)
[16] Stützle, T., Hoos, H.H.: Max-min ant system. Future Gener. Comput. Syst. 16(9), 889–914 (Jun 2000)
[17] Tsutsui, S., Fujimoto, N.: Parallel ant colony optimization algorithm on a multi-core processor. In: Proceedings of the 7th International Conference on Swarm Intelligence. pp. 488–495. ANTS'10, Springer-Verlag, Berlin, Heidelberg (2010)

A terapia de reminiscência em Portugal: oportunidades para ferramentas de suporte digital

Ricardo Antunes[1], Berta Alves[1], Wilmax Cruz[2], Seiji Isotani[2], Luís Carriço[1], and Tiago Guerreiro[1]

[1] LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal
[2] Department of Computer Science, University of Sao Paulo, Brazil

Resumo A doença de Alzheimer afeta cerca de 1% da população portuguesa e duplicará nas próximas décadas. É uma doença crónica e incurável cujo tratamento farmacológico só desacelera a sua evolução, podendo ser complementado com atividades de estimulação. A estimulação cognitiva, nomeadamente a terapia de reminiscência, e a orientação para a realidade diminuem a perda de noção espácio-temporal e o declínio social do doente. No entanto, estas atividades são ainda manuais e analógicas, com materiais de suporte pouco dinâmicos e não personalizados.

De forma a ter uma caracterização dos processos, hábitos e necessidades atuais neste contexto realizámos entrevistas semi-estruturadas a 9 profissionais de saúde de diferentes áreas e um questionário online a 536 cuidadores. Desta forma recolhemos necessidades, problemas e potenciais impactos da utilização de ferramentas tecnológicas no seu dia-a-dia.

As entrevistas revelaram que os cuidados se centram na pessoa, com exercícios de estimulação cognitiva personalizados. É necessário promover a comunicação e tal consegue-se maioritariamente através do uso de fotografias. Para estimular a memória é necessário contextualizá-las - onde, quando e quem - ajudando também à orientação espácio-temporal. No entanto, não existem ferramentas de suporte adequadas para além de ser difícil obter esses materiais junto dos cuidadores. Na perspectiva destes últimos, os questionários mostram um isolamento em casa do binómio cuidador-pessoa, repetindo-se a falta de suporte para atividades estimuladoras. As fotografias são o único material utilizado em larga escala mas somente em suporte analógico, sem integração com qualquer informação que seria relevante. Os cuidadores mostram-se desejosos por novas atividades disponíveis, vendo melhorada a sua qualidade-de-vida.

De acordo com os resultados obtidos desenvolvemos uma platforma que permite fazer reminiscência personalizada, baseada em fotografias e contextualizada com pessoas e locais. O material é recolhido através do perfil de Facebook da pessoa com demência e de estimulação do friendsourcing, distribuindo a carga por toda a rede de amigos e familiares.

Keywords: Demência, Alzheimer, Redes sociais, Friendsourcing, Estimulação cognitiva, Reminiscência

1 Introdução

A doença de Alzheimer, a forma mais comum de demência, afeta atualmente 44 milhões de pessoas e espera-se que o número cresça acima dos 132 milhões em 2050 [1]. Apesar deste crescimento, o apoio fornecido a estas pessoas não está a aumentar na mesma proporção, levando a que estas acabem a ficar aos cuidados de cuidadores informais. Sendo normalmente um elemento da família mais próxima, estes cuidadores acabam a sacrificar o seu bem-estar mental e a sua vida em prol da pessoa com Alzheimer, vendo-se eles próprios reféns da doença.

Ao entrevistarmos 9 profissionais de saúde relacionados com a doença e através de um questionário feito a 536 cuidadores, percebemos as principais dificuldades das partes envolvidas e quais as suas necessidades atuais. Tratando-se de uma doença crónica e sem tratamento, a abordagem resume-se a atrasar a sua evolução, sendo para isso fundamental evitar o isolamento do mundo envolvente, ao mesmo tempo que se estimula a sua memória. A falta de comunição e de diálogo com os outros leva a uma espiral recessiva: a falta de orientação quer a nível de espaço quer de tempo leva ao seu declínio cognitivo, piorando progressivamente até que fique completamente presa no seu próprio mundo.

Atualmente, a estimulação cognitiva é pouco dinâmica e eficiente, com exercícios feitos em papel e lápis, para além de uma constante falta de material de suporte, como fotografias ou músicas significativas. Tendo em conta uma abordagem centrada no doente, desenvolvemos uma plataforma colaborativa de recolha e apresentação de material para terapia de reminiscência, que consiste na recordação do passado através de fotografias, músicas ou outro material, onde todos podem usufruir e ajudar no bem-estar da pessoa com demência. Ao apresentarmos eventos de vida enriquecidos com elementos úteis - como os participantes ou os locais - é possível chegar a uma estimulação mais eficaz das memórias recentes e passadas, potenciando a obtenção de melhores resultados e uma maior motivação para fazer este tipo de exercícios.

2 Trabalho relacionado

Este trabalho insere-se num contexto onde há falta de ferramentas digitais relacionadas com a doença de Alzheimer, especificamente para suporte à terapia de reminiscência. Sendo que estas atividades implicam o uso de material pessoal, iremos usar as redes sociais para agregar todos os que podem contribuir numa só plataforma, facilitando o dia-a-dia dos cuidadores, para além de unir toda a rede em volta de uma causa comum: a melhoria da qualidade de vida da pessoa com demência. Nesta secção mostram-se as ferramentas já existentes no suporte à reminiscência, assim como a potencial relevância do friendsourcing neste contexto.

2.1 Suporte à Terapia de Reminiscência

O 'Multimedia Biographies' [2] é um projecto que permite ao cuidador fazer uma recolha de fotografias representantes dos vários capítulos de vida de uma

pessoa com demência. Essas fotos são depois mostradas numa moldura digital que está sempre ligada numa divisão da casa do paciente, fazendo com que este reveja de forma intrínseca essas memórias antigas. Com uma perspectiva mais imersiva temos o 'IVIRAGE' [6], uma ferramenta que cria um ambiente virtual 3D com base em fotografias de lugares significativos, simulando a presença naquele local. Isto torna-se especialmente útil para estimular memórias sobre locais que tenham sido destruídos pela guerra ou que já tenham passado por demasiadas modificações para que possam ser recordados convenientemente.

Para lá das fotografias existe também um estudo que aborda a utilização de vídeos do YouTube para reminiscência em grupo [10]. Com cada vez mais vídeos da década de 80 e 90 disponíveis, é fácil encontrar eventos históricos ou desportivos marcantes, permitindo reavivar memórias do passado. Para além disso recordar através da música permite estimular partes do cérebro que não seriam exercitadas com outro tipo de material.

Por fim, o 'Portrait' [11] introduz um conceito diferente, sendo uma ferramenta para ser usada em lares e centros-de-dia. Antes de cuidar de um certo paciente, o auxiliar passa por um tablet que está disponível numa zona comum e seleciona a pessoa de que está prestes a cuidar. Aí são-lhe apresentadas várias opções, desde ver um resumo da história de vida até aos interesses ou locais significativos. Agora esse auxiliar tem um tema de conversa que pode introduzir à pessoa enquanto faz as suas tarefas de cuidados de saúde, estimulando as suas memórias e a sua comunicação simultaneamente.

No entanto, o problema transversal a estas soluções está na recolha de material. Para além de recair sempre numa só pessoa - o cuidador principal - é bastante desgastante para este, sendo uma tarefa que exige muito tempo e demora meses a ser concluída [2]. Para além disso, sendo só uma pessoa a contribuir nem sempre se consegue obter tanto material quanto desejado.

2.2 Friendsourcing

O Friendsourcing é uma especificação do crowdsourcing que está vocacionada para a recolha de informação proveniente de fontes fidedignas, pertencentes a um grupo fechado e onde os participantes já possuem conexões sociais entre si [4]. Tendo em conta que esta será uma nova forma para informação pessoal, não faria sentido pedi-la a desconhecidos como é feito no crowdsourcing. Ao utilizar uma rede como o Facebook este micro-voluntariado toma um carácter pessoal, pedindo ajuda àqueles que conhecem e se preocupam com a pessoa com demência. Uma vez que essa ajuda pode ser dada online, independentemente da distância que os separa dessa pessoa, toda a rede está mais motivada a ajudar, mostrando empatia e solidariedade com a causa, aumentando as possibilidades de respostas positivas aos pedidos feitos [5].

Fazer este tipo de pedidos em redes sociais para ajudar pessoas com demência já foi tentado numa perspectiva mais textual [8], pretendendo-se usar a mesma noção neste caso para recolher material para a terapia de reminiscência, como fotografias pessoais ou eventos de vida significativos.

3 Estudos formativos

Para que conseguíssemos recolher quais as necessidades e problemas atuais dos envolvidos com a doença de Alzheimer, começámos por entrevistar 9 profissionais de saúde de diferentes áreas. A sua heterogeneidade permitiu obter uma visão sistémica da questão, permitindo saber que perguntas queríamos fazer posteriormente aos cuidadores. Para tal elaborámos um questionário online a cuidadores informais, inquirindo de forma mais específica sobre os seus hábitos diários com a pessoa com demência.

3.1 Entrevistas a profissionais de saúde

3.1.1 Participantes

O grupo de entrevistados é composto por 4 neuropsicólogos [NP1-NP4], 3 psicológos [P1-P3], 1 enfermeiro [E] e 1 terapeuta ocupacional [TO], todos eles - exceptuando o enfermeiro - com experiência com pessoas com Alzheimer's entre 2 e 10 anos. Estes profissionais trabalham na zona da grande Lisboa, ilha da Madeira e na região do Porto, cada um num contexto diferente, englobando consultórios privados, centros-de-dia e lares especializados.

3.1.2 Procedimento

8 das 9 entrevistas foram realizadas pessoalmente na zona de Lisboa por elementos da equipa deste projeto, por norma no local de trabalho do entrevistado. As entrevistas semi-estruturadas tiveram uma duração aproximada de 60 minutos, tendo como temas globais os hábitos durante as consultas, o tipo de estimulação cognitiva que realizam e que ferramentas utilizam para tal. As perguntas foram dirigidas de acordo com as áreas de maior interesse do entrevistado, potenciando a tal diferenciação entre os diferentes profissionais, tendo sido gravadas em registo áudio com termo de consentimento. A restante entrevista foi feita via e-mail devido à distância do profissional.

3.1.3 Análise

Com base nos registos áudio as intervenções dos participantes foram transcritas e depois codificadas. A codificação [7] de cada entrevista foi feita individualmente por 3 elementos da equipa, intersectando depois as diferentes codificações de forma a chegar a uma opinião agregadora. Tal permitiu chegar a um livro de códigos com diferentes temas e noções, revelando as necessidades e preocupações mais comuns e transversais aos diferentes entrevistados.

3.1.4 Resultados

Os **cuidados centrados na pessoa** foram porventura a noção mais importante retirada do processo, tendo sido referida transversalmente em todas as entrevistas. Todo o trabalho feito quer por cuidadores quer por profissionais deve ter em vista o bem-estar da pessoa com demência, devendo tudo ser personalizado tanto

quanto possível para corresponder aos seus gostos e interesses. "Alguém que goste de flores, mesmo que já não o consiga verbalizar, vai ter sempre prazer em resolver atividades relacionadas com o assunto" [NP1]. Para tal é necessário conhecer a história que está para lá da pessoa enquanto doente, saber quem foi, qual a sua estrutura familiar ou que capacidades ainda mantém. Um bom conhecimento da pessoa enquanto tal e não enquanto doente permite aos profissionais guiar melhor o seu acompanhamento para ir de encontro às suas potencialidades.

Uma das formas a que os profissionais recorrem para impedir o avanço da doença é a estimulação cognitiva, nomeadamente a terapia de reminiscência. Este tipo de atividades é maioritamente feita com recurso a fotografias, fazendo a pessoa falar sobre o que estas a fazem lembrar, ou objetos, exercitando a memória procedimental. Existiram algumas opiniões discordantes acerca do tipo de fotografias a utilizar, se recentes ou antigas. Apesar das memórias recentes serem as primeiras a desaparecer, há profissionais que defendem o uso de fotografias atuais quando a doença ainda está numa fase inicial, cabendo ao profissional lidar com a situação quando o doente não consegue relacionar-se com a foto.

A recolha de material pessoal necessário para a terapia de reminiscência é um processo difícil. Estando por norma a cargo do cuidador principal, este nem sempre está disposto ou disponível para colaborar, mostrando inclusivé desconfiança dos profissionais no que toca a fornecer-lhe as suas fotografias pessoais. Para além desse medo de ver perdidas as suas memórias fotográficas, esta tarefa vai aumentar ainda mais a sobrecarga do cuidador, levando ao seu desinteresse em contribuir.

Para lá de fotografias e objetos é também referida, ainda que de forma mais esporádica, o uso de música, que segundo os profissionais permite estimular partes específicas do cérebro que só são passíveis de serem estimuladas através desta. Já relativamente ao uso de vídeos estes não mostraram entusiamo, não sendo uma ferramenta usada no seu dia-a-dia.

As pessoas com Alzheimer precisam igualmente de exercitar a sua orientação espacio-temporal para não perder noção da realidade. Os resultados desse tipo de exercicio podem ser tanto melhores quanto mais personalizados forem os materiais disponíveis, sendo mais fácil associar memórias a espaços temporais se estas forem apresentadas contextualizadas e relacionadas cronologicamente. A nível espacial é também importante associar os locais em que estas aconteceram, podendo ainda ser completamentadas, para benefício comum, com aqueles que fizeram parte integrante delas, como amigos e familiares. P2 referiu-se a essa perda de noção com a seguinte metáfora: "A memória é como uma mapa dos distritos de Portugal, quando estão todos juntos sabemos dizer que é Portugal mas ao longo do tempo eles vão se separando uns dos outros até acabarmos a não conseguir identificar o todo como Portugal."

A falta de conhecimento dos auxiliares dos centros-de-dia e lares acerca das pessoas de que cuidam é um outro problema identificado pelos profissionais. Ao não saberem quem foi a pessoa de quem estão a cuidar nunca conseguirão relacionar-se com ela, sendo somente alguém que precisa de cuidados médicos. Desta forma não se consegue motivar as pessoas a comunicar: elas precisam de

sentir que há uma rede que se preocupa com elas, alguém que conhece os seus interesses e hábitos.

Esta troca de informação também pode ser interessante no sentido inverso, onde a instituição envia para os cuidadores e familiares fotografias e outros materiais relacionados com eventos do dia-a-dia, permitindo que todos se aproximem e se lembrem mais do ente-querido que têm num lar.

A privacidade, ou falta dela, é apontada pelo pessoal médico como potencial problema, dado que a personalização dos cuidados implica a circulação de informação pessoal e de alguma forma sensível. Existem dois grandes aspetos a ter em conta: a privacidade da pessoa com demência e a privacidade da família. Se em relação à última a decisão recai em quem decide partilhar, ou não, esta informação biográfica, a primeira não é assim tão direta. Existindo um vazio legal relativo a quem tem a responsabilidade, impera o bom senso e a capacidade de julgar se a pessoa com demência ainda tem plenitude de razão para tomar decisões sobre ela própria, podendo assim ser responsável pela partilha dos detalhes da sua vida com aqueles que cuidam dela.

3.2 Questionário a cuidadores

Para complementar esta perspetiva profissional quisemos saber a posição de quem está do outro lado. Realizámos então um questionário online a 536 cuidadores portugueses de pessoas com demência, permitindo assim conhecer os seus hábitos e atividades diárias, que tipo de estimulação cognitiva já efetuam e quais as dificuldades relacionadas com esta.

3.2.1 Procedimento

O questionário foi disponibilizado online através da plataforma Google Forms durante duas semanas, tendo sido divulgado maioritariamente no Facebook através das páginas da Alzheimer Portugal[3] e do Café Memória[4]. Foi também distribuido pela rede de contactos dos profissionais de saúde entrevistados e na mailing-list do Grupo de Suporte a Cuidadores da Alzheimer Portugal.

As perguntas focaram-se em caracterizar o cuidador, a pessoa com Alzheimer e a respectiva rede de suporte, assim como as atividades que fazem juntos. Foram também inquiridos acerca de hábitos de estimulação cognitiva, que materiais utilizam e como é o seu contacto com os profissionais de saúde. No final de cada secção estava disponível uma caixa de comentários onde o cuidador pôde acrescentar informação que achasse relevante e que não tivesse sido coberta pelas perguntas.

3.2.2 Resultados

Caraterizando o cuidador típico português, este é do sexo feminino (86.7%), está empregado (54.8%) e tem pelo menos o ensino secundário (80.8%).

[3] https://www.facebook.com/alzheimerportugal.org
[4] https://www.facebook.com/cafememoriapt

Esta alta taxa de escolaridade pode ser resultado de ser um questionário online partilhado em páginas relacionadas com Alzheimer, levando a que grande parte dos inquiridos sejam pessoas pro-activas em busca de mais informação sobre a doença.

A pessoa com demência de quem cuidam é por norma o seu progenitor (53.6%), sendo o cônjuge o segundo caso mais comum (13.6%), e a sua rede de suporte é maioritariamente composta por 1 a 3 pessoas (86.4%). 71.4% dos cuidadores têm a cargo uma pessoa do sexo feminino entre 71 a 90 anos (80%) que ainda vive em casa (75%). O grau de independência é baixo, com 73.9% a precisar de ajuda para realizar a sua higiene diária, 82.8% incapazes de sair de casa sozinhos e 82.8% de gerir o seu próprio dinheiro.

Relativamente às suas rotinas conjuntas, dar um passeio é a atividade mais efetuada em conjunto, com 56.4% a fazê-lo pelo menos semanalmente. Em segundo lugar temos o convívio com amigos, com 40.6% também a fazê-lo pelo menos uma vez por semana. No que toca a atividades de reminiscência, 31.9% vêem fotos antigas menos que uma vez por semana enquanto 33.7% fazem-no mais frequentemente. A percentagem é similar para a visita a lugares significativos, com 30.4% e 29.6% para as mesmas frequências, respetivamente.

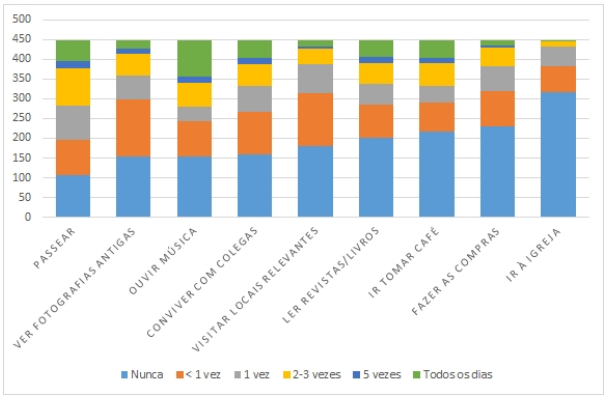


Figura 1. Gráfico representativo dos hábitos diários dos cuidadores com as pessoas de quem cuidam (448 cuidadores responderam a esta secção)

Com uma relevância mais moderada temos a utilização de música: apesar de 20.5% fazê-lo diariamente, só 25% o faz pelo menos uma vez por semana. Um pouco mais alta é a taxa para a leitura de livros e revistas, com 36.2% a fazê-lo semanalmente no mínimo.

É possível verificar que, apesar das diferentes frequências, as atividades mais transversais são passear, ver fotografias antigas e ouvir música. Há no entanto um número considerável de cuidadores que não faz qualquer tipo de atividade fora de casa, com 23.9%, 35.5% e 40% responderam 'nunca' sobre passear, conviver

com amigos e visitar lugares significativos, respetivamente. Essas percentagens sobem ainda mais nas idas ao café, às compras ou visitas à igreja.

Quando inquiridos acerca de hábitos de estimulação cognitiva, os cuidadores dividem-se, com 54.7% a respondem afirmativamente. 38.4% fazem estas atividades diariamente e 30.6% 2 a 3 vezes por semana. A forma mais frequente é o uso de papel e lápis, com 60%, seguindo-se livros de exercícios como palavras-cruzadas ou sudoku, com 28.2%. Já ferramentas em suporte digital só são utilizadas por 17.1% da amostra analisada.

No que toca especificamente à reminiscência, 75.9% revê memórias de alguma forma, nomeadamente fotografias e música. Essa percentagem sobe para 89.7% quando perguntados se essa visita ao passado incide sobre eventos de vida, como o dia do casamento ou o nascimento de um filho. Com 87.1%, as fotografias pessoais revelam-se como o material mais usado para a reminiscência, seguindo-se de objetos significativos e de música, com 31.2% e 30.9%. Fotografias genéricas e vídeos são usadas somente por uma percentagem residual dos inquiridos, sendo os vídeos online o material menos popular com apenas 6.2% a identificá-los como algo que utilizam.

A forma de recolha destes materiais foi também alvo de perguntas, revelando que esta atividade é da única responsabilidade do cuidador: 71.8% revelam que nunca pedem a ajuda de terceiros para esta recolha. Para lá dos materiais é importante perceber os temas abordados na estimulação. 87.6% dos cuidadores dizem que os melhores resultados são alcançados quando falam sobre as pessoas que estão nas fotografias apresentadas.

Dentro dos que não fazem estimulação cognitiva, o motivo mais comum, com 56.2%, é a falta de interesse da pessoa com demência, seguindo-se a falta de tempo - 23.2% - e a inexistência de material de suporte, com 22.2%. Nalguns dos casos foi reportado na caixa de comentários que o cuidador é ele próprio demasiado idoso para ter este tipo de iniciativas.

As relações com as instituições de acolhimento foram também alvo de análise neste questionário. Os cuidadores que têm os seus familiares institucionalizados não foram perguntados acerca das rotinas e dos exercícios de estimulação, focando-se antes nesta relação e nas trocas de informação existentes. Numa escala de 1 a 5, sendo 5 o maior grau de satisfação, 33.3% classificaram a quantidade de informação que recebem das instituições no nível 3. Globalmente a satisfação é positiva, com apenas 19.5% das respostas abaixo desse nível.

No entanto, 74.4% dizem que essa informação só lhes é transmitida quando requisitada, não sendo costume as instituições serem pró-ativas. Acerca do conhecimento dos profissionais acerca dos seus familiares, 50.6% acham que estes não têm conhecimento suficiente da sua história de vida ou interesses.

A frequência de contacto com os profissionais é bastante distinta entre classes, sendo apenas o neurologista visto mais que uma vez ao ano pela quase maioria dos cuidadores (47.5%). As taxas de resposta para 'nunca' são altamente expressivas para neuropsicólogos - 80.7% - psicólogos - 77.2% - e terapeutas ocupacionais - 76.4%. Um caso especial acontece nos enfermeiros: se por um lado 20.9% têm um contacto semanal, 42.8% voltam a responder que o contacto é inexistente.

Por fim, a ligação e utilidade das redes sociais relativamente à doença revela que 49.2% faz parte de algum grupo de discussão relacionado com Alzheimer. No entanto apenas 21.9% as usa para partilhar fotos da pessoa com demência com o resto da família. No sentido inverso os resultados sobem ligeiramente, com 35.5% a responderem afirmativamente à utilização de redes sociais para apresentarem material para terapia de reminiscência. À cerca da potencial contribuição das redes sociais para aproximar família, cuidadores e profissionais, 32.7% responderam com a nota máxima, correspondente ao nível 5. Novamente numa escala de 1 a 5, 79.7% responderam 3 ou mais elevado à mesma pergunta.

4 Implicações para o sistema

Tendo em conta que a maior parte dos cuidadores acaba a isolar-se em casa, a forma de atingir e ajudar um maior número será promover atividades que possam ser feitas dentro de portas. Desta forma, as funcionalidades oferecidas serão igualmente úteis para os que ainda saem de casa e para os que já não o fazem.

4.1 Personalização de conteúdos

Para obtermos melhores resultados percebemos que a estimulação cognitiva, mais especificamente a terapia de reminiscência, tem de ser personalizada para cada doente. Tal só pode ser alcançado com material pessoal e depois de saber os gostos e interesses dos pacientes. No fundo, é preciso fazer que quem cuide delas as conheça enquanto pessoa, sabendo como as motivar a fazer este tipo de exercícios [9].

Para o uso deste material pessoal é necessário uma recolha prévia do mesmo, estando aqui um dos problemas identificados. Esta pesquisa e recolha precisa de ser agilizada, distribuindo a carga por várias pessoas, evitando que fique ao cargo de uma só. O objetivo primordial é tornar o processo menos moroso e mais motivador para todos. Fotografias, músicas e objetos são os materiais mais utilizados, devendo esta optimização da recolha incidir sobre os mesmos. Ao utilizar ferramentas digitais é atualmente dificil lidar com a terceira dimensão pelo que o foco se ficará, por agora, nas fotografias e músicas. No entanto, nada impede que a reminiscência com base em objetos seja feita de forma indireta, através de fotografias dos mesmos.

4.2 Enriquecer a estimulação

Conjugando o que seria melhor para a saúde do doente e o que seria mais estimulante para o cuidador, julgamos que a apresentação do material de reminiscência deve ser feito na forma de eventos de vida. Estes eventos permitem a contextualização temporal, assim como a identificação dos participantes, referida como necessidade pelos profissionais. Para além disso, os cuidadores reportam que

os melhores resultados são alcançados quando fazemos a pessoa com demência falar sobre as pessoas que conhece, algo que podia ser potencializado se cada evento mostrasse as fotos e nomes de quem esteve presente, ajudando a reavivar a memória [3]. Esta solução permite ainda manter a abordagem que os cuidadores têm atualmente, muitas vezes baseada em rever esses eventos de forma isolada, fazendo com que a transição para o meio digital seja mais natural.

4.3 Novas pontes de comunicação

Por fim, é também necessário não esquecer aqueles cujos familiares já estão institucionalizados. Indo de encontro à preocupação encontrada em ambas as partes - cuidadores e profissionais - é preciso fazer com que os profissionais de saúde tenham acesso à informação biográfica que lhes permita conhecer os seus pacientes. Mais ainda, esta troca de informação pode também ser relevante para a comunicação instituição-família, permitindo que esta acompanhe de forma mais imediata o que se passa na instituição. Considerando estas atividades também como eventos de vida, cria-se um album bi-direcional da vida da pessoa, fortacelendo os laços entre os membros da família e a pessoa com demência.

5 Plataforma de Suporte à Terapia de Reminiscência

Adereçando o conjunto de problemas alcançados com a recolha de requisitos chega-se à nossa plataforma integradora, acoplando vários módulos direcionados a cada uma das partes envolvidas. Esta aplicação está desenvolvida e pronta a ser disponibilizada ao público, primeiramente numa fase de testes e validação e de seguida sem qualquer restrição. O objetivo primordial é melhorar a terapia de reminiscência através da facilitação da recolha de materiais, uma maior contextualização dos mesmos e também de uma acessibilidade mais generalizada a todos os interessados. Espera-se que este dinamismo leve a uma maior motivação por parte dos cuidadores e também dos doentes de Alzheimer's, captando a sua atenção através de materiais mais interativos e funcionais.

Esta aplicação web é desenvolvida com responsive-design, sendo assim corretamente apresentada em qualquer dispositivo, independentemente do sistema operativo ou dimensão do ecrã, bastando acesso à internet para a utilizar. Baseando-se num perfil de Facebook criado de propósito para a pessoa com demência, a aplicação vai recolher e compreender que materiais são relevantes para essa pessoa. Esses materiais vão ser contextualizados e interactivos, ficando depois disponíveis para serem utilizados pelo cuidador como material para sessões de reminiscência. Para casos em que a pessoa já não esteja em casa, a aplicação passa a ser um veículo de transmissão das atividades relevantes que são feitas no lar ou centro-de-dia para os seus familiares, assim como um repositório de interesses e história de vida da pessoa com demência.

5.1 Arquitectura do sistema

Cada vez mais ligados às redes sociais, a sua utilização para enriquecer aplicações de terceiros tem-se tornado num processo comum. Ao evitar que o utilizador tenha de aprender a interagir com uma nova plataforma estamos a mitigar eventuais problemas, apresentando-lhe uma interface já conhecida - o Facebook. Assim, pretende-se que o cuidador principal crie, sendo administrador e responsável, um perfil para a pessoa com demência. Este será preenchido com os dados biográficos básicos e uma fotografia de perfil. Deve também dar "gosto" em páginas que sejam relevantes para a pessoa com demência, onde cada "gosto" será interpretado pelo sistema como um interesse.

Posto este passo inicial, de forma a que se torne efetivamente uma rede, deverá convidar amigos que conheçam a pessoa com demência. Estes amigos poderão depois ser convidados a participar em eventos – a base da recolha de dados do nosso sistema. Os eventos do Facebook funcionarão como eventos de vida: ao criar um novo, o cuidador deve definir o nome, p.e. Casamento, indicando o local onde aconteceu, a data, convidar amigos que estiveram presentes e adicionar uma foto de capa, caso tenha alguma relativa a esse evento, conforme se pode observar na figura 2.

Assim que os convidados aceitem o convite todos eles podem contribuir com material, bastando para isso fazer o upload das memórias que tenham para o mural do evento. Desta forma distribui-se a recolha do material por todos os que tenham participado nesse evento e queiram ajudar à elaboração deste álbum de vida. Repetindo esta ação para tantos eventos quantos deseje, o perfil vai ficar enriquecido com um conjunto de memórias que serão depois utilizadas como material para fazer terapia de reminiscência. É também possível participar em eventos que não tenham sido criados pelo cuidador, bastando para isso que este aceite o convite feito por outro amigo pertencente à rede.

Figura 2. Evento de vida - Viagem à Madeira - no Facebook

Esta integração com o Facebook permite também salvaguardar-nos de eventuais problemas legais e de autenticação, ficando o cuidador sujeito às políticas de privacidade da rede social. Estes eventos deverão, à partida, ser privados mas tal estará ao critério de cada cuidador. Toda a informação consumida pela nossa aplicação provém só e unicamente desta fonte, estando previsto numa fase posterior abrir a possibilidade do cuidador inserir eventos diretamente no sistema, sem utilizar o Facebook.

5.2 Aplicações de uso

5.2.1 Personalização dos cuidados clínicos Passando à aplicação propriamente esta disponibiliza duas vistas distintas. Relativamente à primeira, direcionada para auxiliares de lares e centros-de-dia, é objetivo permitir então que aqueles que cuidam diariamente das pessoas institucionalizadas saibam um pouco mais acerca da sua história de vida e dos interesses da pessoa de quem vão cuidar.

Tendo em conta que é mais complicado restringir aqueles que têm acesso a esta informação, para além dos profissionais não terem muito tempo disponível no seu dia-a-dia laboral, esta será reduzida e sintetizada em tópicos. Funcionando à imagem do que acontecia no 'Portrait' [11], esta aplicação estará disponível num tablet fixo e comum a todos os funcionários. Num ecrã inicial onde podem ver todos os pacientes da instituição com perfil na aplicação, eles poderão selecionar o seu próximo paciente e ser-lhes-á apresentado um ecrã com três escolhas: eventos de vida, lugares e interesses.

Figura 3. Excerto da história de vida de Joaquim Torres

Nos eventos de vida (Figura 3) é apresentada a descrição do evento, com nome, data e local, sendo esta a vista mais abrangente de todas. Os lugares representam lugares significativos para a pessoa, sendo um resumo dos vários locais onde aconteceram os eventos de vida. Por fim os interesses apresentam os

"gostos" do perfil da pessoa, permitindo encontrar interesses em comum entre a pessoa com demência e o profissional que está prestes a cuidar dela. Em suma, queremos dar tema de conversa a esse profissional para estimular o discurso da pessoa com demência.

Frases como "Então ouvi dizer que se casou em 1975 na França" farão eventualmente a pessoa falar sobre algo que se recorde. A mesma abordagem pode ser feita com eventos mais recentes, especialmente em clima de centro-de-dia: "Então onde foi com a sua filha na semana passada?" sabendo o profissional à partida que foi uma viagem à Madeira, por exemplo. Este estímulo ao diálogo pode também ser feito com base nos interesses, encontrando pontos em comum às partes que de outra forma seriam difíceis de descobrir.

5.2.2 Ferramenta para suporte à terapia de reminiscência A segunda vista da aplicação é feita para ser utilizada quer em casa quer em clima de consulta de acompanhamento por neuropsicólogo, psicólogo ou terapeuta ocupacional. Neste caso não há informação escondida do utilizador, tudo o que é obtido via Facebook é apresentado numa interface clara e simples ao utilizador. Para além de permitir que a pessoa com demência se foque só no essencial - as fotos - eliminando toda a informação desnecessária que apareceria se lhes mostrássemos directamente o evento no Facebook, a aplicação fornece também uma maior contextualização, enriquecendo os eventos com informação não-pessoal proveniente de outras fontes como o Flickr. Desta forma espera-se aumentar a motivação da pessoa com demência, assim como conseguir que esta se mantenha atenta e focada na atividade durante um período maior de tempo.

Ao aceder a esta aplicação web o utilizador terá de fazer login com as credênciais da conta de Facebook da pessoa com demência, previamente criada. Na página inicial (Figura 4) pode ver um resumo de todos os eventos em que esteve, sendo cada um uma hiperligação para os seus detalhes. Estes podem ser apresentados ordenados pela data em que aconteceram ou pelo seu nome.

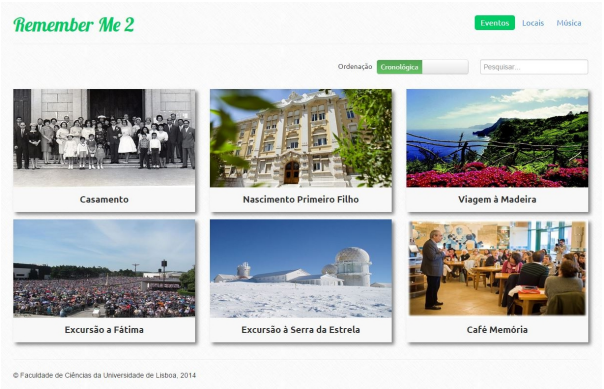


Figura 4. Página inicial mostrando todos os eventos

Ao selecionar um, o utilizador pode ver quem participou no evento - e respetivas fotografias de perfil - quando e onde foi. Isto fornece uma contextualização têmporo-espacial ao foco principal da página do evento: as fotografias recolhidas pelos participantes. São estas que ocupam o lugar de destaque na página, num slideshow navegável e onde se pode fazer zoom, permitindo também focar as caras de quem está nas fotos (Figura 5).

Caso o lar ou centro-de-dia esteja interessado em colocar os seus eventos na plataforma basta também criar uma conta no Facebook e convidar os pacientes que utilizem a aplicação para os seus eventos. Assim que o cuidador responsável pelo perfil da pessoa com demência aceite o convite, esse evento aparecerá na aplicação, podendo ver as fotografias que a instituição ou outros participantes lá tenham colocado.

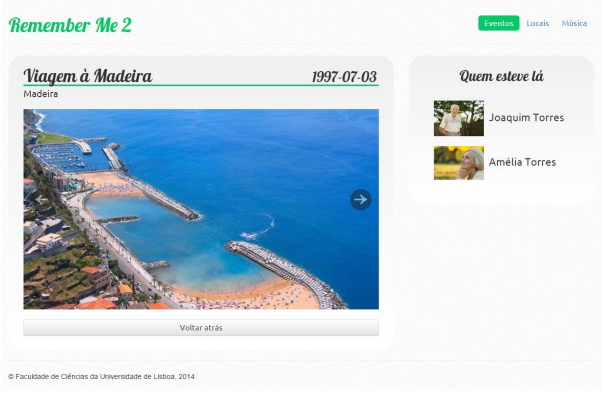


Figura 5. Detalhe do mesmo evento - Viagem à Madeira - na aplicação Remember-Me

A partir de cada evento podemos também aceder a mais fotos, desta vez não pessoais, mas do local onde o evento ocorreu. Bastando carregar por cima do nome do local somos transportados à respetiva página, onde a aplicação recolhe e apresenta fotografias provenientes do Flickr sobre esse mesmo local, tendo em consideração a data em que o evento ocorreu. Imaginando que um certo evento ocorreu no Rossio em Lisboa no ano de 1990, a aplicação faz a pesquisa por fotos tiradas entre 1985 e 1995 nesse mesmo local. Isto permite que a pessoa com demência viaje no tempo e se recorde de locais significativos na altura em que o foram. Permitindo uma maior contextualização, são também apresentados os amigos que estão ligados a esse local, tendo em conta os eventos em que participaram.

É também possível navegar pelos locais da mesma forma que se navega pelos eventos, bastando para isso aceder à página respetiva através do menu. Aí aparecerão todos os eventos, podendo mais uma vez serem ordenados quer cronologicamente quer alfabeticamente, bastando ao cuidador escolher sobre qual deseja reminiscer da mesma forma que o faz na página de eventos.

Por fim, tendo em conta os materiais mais utilizados pelos cuidadores, a aplicação oferece a opção 'Música'. Apresentando um menu à esquerda composto pelos interesses musicais obtidos através dos gostos que o perfil tem no Facebook, o utilizador é convidado a selecionar uma das opções desse menu para que lhe seja disponibilizada automaticamente uma playlist com músicas do respetivo artista provenientes do YouTube, poupando tempo na pesquisa e aumentando a versatilidade do sistema.

6 Conclusão

O objetivo primordial desta investigação está na percepção do caminho a seguir para facilitar o dia-a-dia daqueles que têm de lidar com pessoas com Alzheimer através de soluções tecnológicas. Percebeu-se que as necessidades dos profissionais e dos cuidadores se interesectam em determinados pontos: é preciso combater a fraca motivação para a estimulação cognitiva, tornar os materiais para a mesma mais interactivos e especializados e olhar para o doente como a pessoa que foi no passado. No entanto há também necessidades específicas em cada grupo que não devem ser desprezadas. Por um lado os profissionais revelaram uma falta de material de trabalho nas consultas de acompanhamento, assim como da falta de conhecimento dos auxiliares dos lares e centros-de-dia sobre os seus pacientes. Na perspectiva dos cuidadores estes precisam de mais apoios para o seu dia-a-dia, procurando alternativas mais eficientes para as suas tarefas diárias.

Esperamos que esta pesquisa possa abrir novos caminhos para o futuro, quer no desenvolvimento de novas plataformas como na melhoria desta, nomeadamente a inclusão da reminiscência através de objetos através de impressoras tridimensionais.

Agradecimentos Agradecemos à Alzheimer Portugal pelo apoio prestado e acesso à sua rede de contactos de profissionais e cuidadores. Agradece-se também à Fundação Para a Ciência e a Tecnologia (unidade de inverstigação LaSIGE, ref. UID/CEC/00408/2013), à FAPESP (Process: 2013/13056-4 and 2014/12002-0) e CNPq (Processes: 310204/2011-9; 400481/2013-8 and 470757/2013-2) por darem suporte a este projeto.

Referências

1. World alzheimer report 2014. https://www.alz.co.uk/research/WorldAlzheimerReport2014.pdf, accessed on 03-06-2015
2. Baecker, R.M., Marziali, E., Chatland, S., Easley, K., Crete, M., Yeung, M.: Multimedia biographies for individuals with alzheimer's disease and their families. In: Proceedings of the World Congress on Internet in Medicine (MEDNET) (2006)
3. Barco, C., Yatani, K., Ma, Y., Pal, J.: Communication and coordination for institutional dementia caregiving in china. Ann Arbor 1001, 48109 (2013)

4. Bernstein, M.S., Tan, D., Smith, G., Czerwinski, M., Horvitz, E.: Personalization via friendsourcing. ACM Transactions on Computer-Human Interaction (TOCHI) 17(2), 6 (2010)
5. Brady, E., Morris, M.R., Bigham, J.P.: Gauging receptiveness to social microvolunteering. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems. pp. 1055–1064. ACM (2015)
6. Chapoulie, E., Guerchouche, R., Petit, P.D., Chaurasia, G., Robert, P., Drettakis, G.: Reminiscence therapy using image-based rendering in vr. In: Virtual Reality (VR), 2014 iEEE. pp. 45–50. IEEE (2014)
7. Charmaz, K.: Constructing grounded theory. Sage (2014)
8. Martins, J., Carilho, J., Schnell, O., Duarte, C., Couto, F.M., Carriço, L., Guerreiro, T.: Friendsourcing the unmet needs of people with dementia. In: Proceedings of the 11th Web for All Conference. p. 35. ACM (2014)
9. McKeown, J., Clarke, A., Ingleton, C., Ryan, T., Repper, J.: The use of life story work with people with dementia to enhance person-centred care. International Journal of Older People Nursing 5(2), 148–158 (2010)
10. O'Rourke, J., Tobin, F., O'Callaghan, S., Sowman, R., Collins, D.: 'youtube': a useful tool for reminiscence therapy in dementia? Age and ageing 40(6), 742–744 (2011)
11. Webster, G., Fels, D.I., Gowans, G., Hanson, V.L.: Portraits of individuals with dementia: views of care managers. In: Proceedings of the 25th BCS Conference on Human-Computer Interaction. pp. 331–340. British Computer Society (2011)

Exploring APIs with N-gram Language Models

Gonçalo Prendi[1], Hugo Sousa[1], André L. Santos[1], and Ricardo Ribeiro[1,2]

[1] Instituto Universitário de Lisboa (ISCTE-IUL), Av. das Forças Armadas, 1649-026 Lisboa
[2] INESC-ID Lisboa, R. Alves Redol 9, 1000-029 Lisboa
goncalo.prendi25@gmail.com, hugossousa92@gmail.com
andre.santos@iscte.pt, ricardo.ribeiro@iscte.pt

Abstract. Software development requires the use of external Application Programming Interfaces (APIs) in order to reuse libraries and frameworks. Programmers often struggle with unfamiliar APIs. Such difficulties often lead to incorrect sequences of API calls that may not produce the desired outcome. Language models have shown the ability to capture regularities in text as well as in code. In this paper we explore the use of n-gram language models and their ability to capture regularities in APIs. We explored some of the most widely used APIs with the Java programming language, training several language models over hundreds of GitHub Java projects that use these APIs. The evaluation shows perplexity values for these language models that hint the possibility of using them to produce a tool to assist developers with code completion when using an unfamiliar API. Such a tool may help developers to write correct API call sequences more efficiently; moreover, allows them to explore the features offered by the API.

Keywords: APIs, Java, n-gram language models, perplexity, source code mining, code completion

1 Introduction

The use of third-party APIs is inherent to modern software development, since it provides a reliable and efficient way of producing software. The correct use of an API can sometimes be hard to perceive if the provided documentation is insufficient or inadequate. The API design itself may cause discoverability hurdles related to the relationships between API types and the sequence of calls that are necessary to obtain an instance of a certain type. These are some of the difficulties programmers often find when using an unknown API [18, 9, 14]. These difficulties may lead to an incorrect use of an API and slow down the realization of programming tasks.

In order to overcome the API usability problems, extensions to Integrated Development Environments (IDEs) have been developed to assist programmers on their API-related tasks (e.g., [6]). Often these tools use Natural Language Processing (NLP) techniques to create statistical models, in order to recommend

a set of valid API calls. We refer to these sequences as API *sentences*, which are composed of *tokens* from the API *vocabulary*. These *sentences* are usually extracted from source code corpora. The main contribution of this paper is an overview on the performance of n-gram language models [5] in capturing usage patterns in the context of APIs and which techniques best suit each API.

This paper is organized as follows. Section 2 describes the API sentence extraction method. Section 3 gives an overview over the language models and respective smoothing techniques, Section 4 presents the evaluation methodology and results achieved for each of the APIs. Section 5 discusses the results achieved and draws some conclusions for other APIs as well as a description of threats to validity that might influence the results obtained. Section 6 addresses related work. Section 7 presents our conclusions and outlines future work towards improving the models.

2 API Sentences and Tokens

Language models are highly dependent on how the information is represented in order to produce satisfactory recommendations. Therefore, the extraction process of the API sentences from source code corpora is crucial. A poor extraction process will result on an inaccurate or noisy model, thus producing low quality recommendations that might become more of an obstacle than an aid to the programmer. In order to minimize this noise, our extraction process is handled very carefully. We developed a Java parser on top of the Eclipse's Java Development Tools for this purpose. Next, we present an overview of how API tokens and sentences are handled in the extraction process.

2.1 Tokens

A token is a valid API call that involves a public type and a public operation therein. Static methods and constructors are considered operations as well. Currently, operation parameters are not taken into account (which is a possible improvement for future work), therefore method overloading is not handled. Since these tokens are abstractions of the actual code instruction, they are represented by a tuple consisting on the API's type and the operation. Take the following instructions from the SWT[3] API as an example:

```
Label label = new Label(...);
label.setText(...);
```

The first instruction is represented as the token $\langle Label, new \rangle$ and the second as $\langle Label, setText \rangle$. As stated before, constructors are handled as operations, so we decided to use the "new" keyword to differentiate it from the other operations.

[3] https://www.eclipse.org/swt/

2.2 Sentences

An API sentence is a sequence of tokens, representing related instructions. Formally, a sentence ω is a vector of tokens from the API's vocabulary $\mathcal{V}$:

$$\omega = (t_0, t_1, \ldots, t_n) : \forall t \in \omega, t \in \mathcal{V}$$

Take the following code snippet as another example:

```
Composite composite = new Composite(...);
composite.setLayout(new GridLayout(...));
```

The resulting extracted sentence would be:

$$\omega = (\langle Composite, new \rangle, \langle GridLayout, new \rangle, \langle Composite, setLayout \rangle)$$

In this code snippet there are three API calls. The instantiation of the types `Composite` and `GridLayout` and the operation `Composite.setLayout`.

Any instructions that do not represent valid calls on the SWT API are ignored in the parsing process. The tokens' order reflects the execution path, where chained invocations are separated into individual instructions.

Token dependencies are also considered when building the API sentences. Since instructions can be interleaved, an instruction a only depends on another instruction b in the following cases:

1. b is an assignment and a uses the assigned variable;
2. a is an invocation and b is used in its arguments.

The above guarantees that all the dependencies will be represented in the same sentence. Finally, similarly to the MAPO tool [21] (which is used for mining API usage from repositories), `if-else` blocks result in a set of sentences for each possible branch. As for the loop blocks, repetitions are ignored and they are treated as regular selection blocks.

3 API Sentence Models

Language models are widely used on several written and spoken language processing tasks, such as speech recognition [17], spell-checking and correction [15] and machine-translation [4]. Typically, APIs have regular usage patterns that describe valid sequences of API calls. Language models provide a simple and efficient way of capturing these regularities.

3.1 Overview

Language models are statistical models that allow the computation of the probability of a sentence or the estimation of how likely a history of tokens will

be followed by a certain token. These probabilities are described as the product of a set of conditional probabilities. Hence, the probability of a sentence $\omega = (t_0, t_1, \cdots, t_n)$ is given by:

$$P(\omega) = P(t_0)P(t_1|t_0)P(t_2|t_0t_1)\cdots P(t_n|t_0t_1\cdots t_n - 1) \tag{1}$$

Equation 1 represents a chain of conditional probabilities $P(t|h)$, where t is the token and h is the history or the previously written tokens. To compute these probabilities the maximum likelihood estimate is used, where $C()$ is the number of times the sequence appears in the training set, which is given by:

$$P(t_n|t_0t_1\cdots t_n - 1) = \frac{C(t_0t_1\cdots t_n)}{C(t_0t_1\cdots t_n - 1)} \tag{2}$$

Since it is not reliable to estimate probabilities for long histories, it is common to set a limit N for these long histories using the Markov assumption. The probabilities are then computed as follows:

$$P(t_n|t_{n-N+1}\cdots t_{n-1}) = \frac{C(t_{n-N+1}\cdots t_n)}{C(t_{n-N+1}\cdots t_{n-1})} \tag{3}$$

N-gram language models are some of the several existing statistical models and have been previously used [11] to capture regularities in source code in general. Our intuition was that n-gram language models would work well in this context, since APIs usually have common usage patterns or regularities with relatively small histories, which can be captured with these models.

After extracting the API sentences from the source code corpora, the SRILM toolkit [19] was used to build the n-gram language models and apply different smoothing techniques available, as well as to evaluate the models' performance.

3.2 Smoothing techniques

Language models suffer from a problem called data sparseness. This problem especially arises when the corpora is too small and does not contain all possible sentences, which will result in a zero probability for those sentences when querying the model, due to the nature of the maximum likelihood estimate. Smoothing techniques are used to address this problem by producing more accurate probabilities. This is done by adjusting the maximum likelihood estimate of probabilities [7] by attempting to increase the lowest probabilities (such as zero) and decrease the highest ones resulting on a more uniform probability distribution, and hence, producing a more accurate model. The SRILM toolkit supports smoothing techniques that we have experimented in our evaluation, namely the Witten-Bell and Kneser-Ney methods.

Witten-Bell If a given n-gram occurs in the training set, it is reasonable to assume that we should use the highest order n-gram in order to calculate the probability of the next token. The model in this situation will be much more

accurate than using the lower order ones that may recommend a larger set of possible tokens. However, when the n-gram does not appear in the training set, we can back off to the lower order ones. Equation 4 shows how the Witten-Bell smoothing addresses this problem.

$$P_{\mathrm{WB}}(t_i|t_{i-n+1}^{i-1}) = \lambda_{t_{i-n+1}^{i-1}} P(t_i|t_{i-n+1}^{i-1}) + (1 - \lambda_{t_{i-n+1}^{i-1}}) P_{\mathrm{WB}}(t_i|t_{i-n+2}^{i-1}) \quad (4)$$

This equation is based on a linear interpolation between the maximum likelihood estimate of the n-order model and the $(n-1)$-order smoothed model. The weight λ given to the lower order models is proportional to the probability of having an unknown word with the current history (in our experiments, this value was estimated by the SRILM toolkit).

Kneser-Ney A very frequent token t_1 would be represented in the model with a very high unigram probability. However, if this token is only observed after another token t_0 in the training set, it is very unlikely it will appear after another token t_x. A smoothed probability estimate may be high if the unigram probability of t_1 is taken into account when backing-off. Equation 5 shows how the Kneser-Ney [12] smoothing technique addresses this problem, where D is a constant given by Equation 6 ($n1$ and $n2$ are the total number of n-grams with exactly one and two counts) that is subtracted to the n-gram count, when computing the discounted probability and $\gamma(t_{i-n+1}^{i-1})$ is used to make the distribution sum to 1 (for more details on the computation of these values see `http://www.speech.sri.com/projects/srilm/manpages/ngram-discount.7.html`) [8].

$$P_{\mathrm{KN}}(t_i|t_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{C(t_{i-n+1}^{i-1}) - D, 0\}}{\sum_{ti} C(t_{i-n+1}^{i-1})} & \text{if } C(t_{i-n+1}^{i-1}) > 0 \\ \gamma(t_{i-n+1}^{i-1}) P_{\mathrm{KN}}(t_i|t_{i-n+2}^{i-1}) & \text{if } C(t_{i-n+1}^{i-1}) = 0 \end{cases} \quad (5)$$

$$D = n1/(n1 + 2 * n2) \quad (6)$$

This method consists of taking into account the number of different tokens that precede a token t_1 to calculate the unigram probabilities, and not only the number of occurrences of that token.

While learning an unknown API, programmers tend to search for resources in order to complete a task. If most of the programmers use a certain example, that pattern will have a very high probability in the model. However, when executing a more specific task, programmers might use some uncommon operations, that will be reflected in the model with a low probability, since they will occur less often in the source code corpora. A code recommendation tool based on this type of models will keep the programmer on the correct path, but will allow for some exploration of the API by providing some of the uncommon operations.

API	Projects	Sentences	Vocabulary
SWT	501	105,718	4,085
Swing	2,294	160,961	8,105
JFreeChart	248	22,948	3,788
JSoup	115	1,508	370
JDBC Driver	559	1,666	176
Jackson-core	71	9,306	253

Table 1. APIs under analysis: number of client projects, number of extracted sentences, and vocabulary size.

4 Measuring API Perplexity

To evaluate the performance of n-gram language models, we produced a model for each API and measured its perplexity. This measurement is one of the most common methods for evaluating language models. Equation 7 formally describes how the perplexity is computed, where $p(T)$ is the probability assigned to a test set T with a length W_T (number of tokens).

$$P(T) = 2^{-\frac{1}{W_T} log_2 p(T)} \tag{7}$$

This technique provides very useful information about the model, such as the average number of choices for each word [10], which is very important since we want the developer to stay in the right path but also allow for some API exploration. Further, it allows us to determine which N value and which smoothing technique best fits each of the APIs, consequently producing the most accurate model.

4.1 Setup

Allamanis and Sutton [2] collected a large corpus from GitHub, containing thousands of Java projects. To perform our evaluation we selected the projects containing references for each of the APIs from the same GitHub Java Corpus. Table 1 shows the APIs that were evaluated, and the number of projects containing at least one reference to the API's root package, either via import statement or qualified type references. After determining which projects contain these references, we used the method described in Section 2 to extract the API sentences.

The performance evaluation for each API was made with a 5-fold cross validation scheme, using both the Witten-Bell and Kneser-Ney smoothing methods, as well as without any smoothing method, in order to evaluate how differently the model performs[4].

There are some words that may not be found in test sets, which are called Out Of Vocabulary (OOV) words. When the perplexity values are computed,

[4] In SRILM, for each of the smoothing methods, there are 4 different options (no options, `-unk`, `-interpolate` and `-interpolate -unk`).

these words are not taken into account. For this we need to use a special token, `<unk>`, representing the OOV words.

In our case, the interpolate option is used only when using a smoothing method, Witten-Bell and Kneser-Ney. This option interpolates the n-order estimates with the lower order ones, which, in general, are more accurate.

After computing the results, we concluded that the interpolation option alone, produces better results in 80% of the cases. The baseline was the perplexity without any smoothing methods, compared with the Witten-Bell and Kneser-Ney smoothing methods with the interpolation option. Also, we do not show the results for unigrams since they do not support the existence of an history when estimating the probabilities of the next token. The results presented are the averages of the 5-fold cross validation scheme.

4.2 SWT API

SWT is a cross-platform library to develop graphical user interfaces. There are several examples on the web on how to interact with its API. Even though the interface is application-specific and might differ a lot from one another, some of the method calls must always be performed in order to correctly build and customize the user interface.

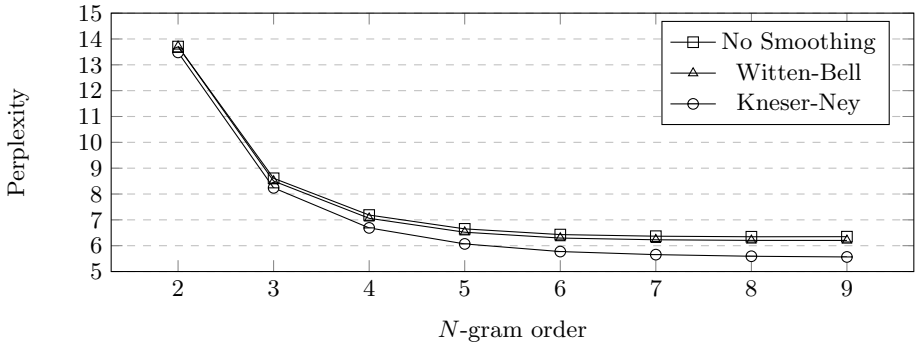


Fig. 1. Perplexity values for the SWT API.

Figure 1 shows that as we increase the value of n, the perplexity decreases and stabilizes at $n = 6$. Thereon it decreases less than 0.2 in total. We can also see that the Kneser-Ney smoothing performs slightly better than all the others. Although the SWT library is used to produce graphical user interfaces, it shows a relatively low perplexity value, with an average of 6 different recommendations for each history of length 5 or higher.

4.3 Swing API

Similarly to the SWT, the Swing library is also used to create graphical user interfaces, however it produces a more similar appearance independently of the operating system in use. Again, the graphical user interfaces are very customizable, and there are several different widgets and components the developer can use, which produces a larger number of recommendations for the same history.

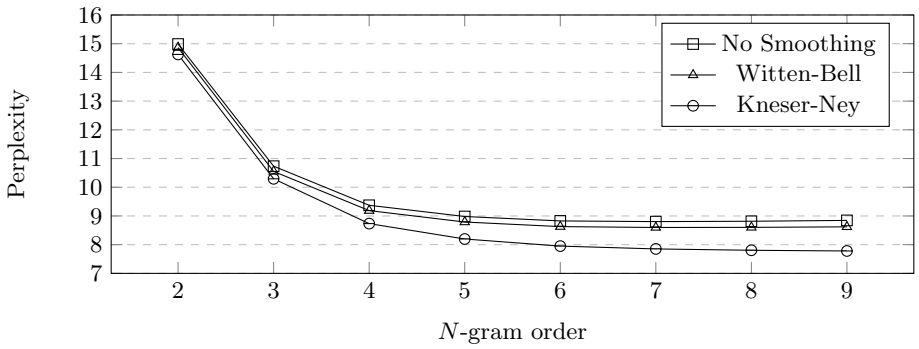


Fig. 2. Perplexity values for the Swing API.

Regarding the objective, it is reasonable to assume that the Swing API is similar to the SWT API. This API may be used for the same domain as the previous one, and although the perplexity values are somehow similar (see Figure 2), they are slightly higher. Nevertheless, the values stabilize at n=6, and the Kneser-Ney smoothing is also the best method as with the SWT API.

4.4 JFreeChart API

The JFreeChart library[5] allows developers to create and customize different kinds of charts in a graphical application. Although the library allows programmers to customize charts, it is not as flexible as the previous ones.

As expected, and since the JFreeChart API is much more regular and has a lot less different paths to follow on each word, the perplexity values are much lower than the ones obtained on the previous APIs. Figure 3 shows that the perplexity values range from 1.97 to 2.34. Although the Kneser-Ney smoothing still achieves the best results, in this case the difference is not significant in comparison with the Witten-Bell method.

[5] http://www.jfree.org/

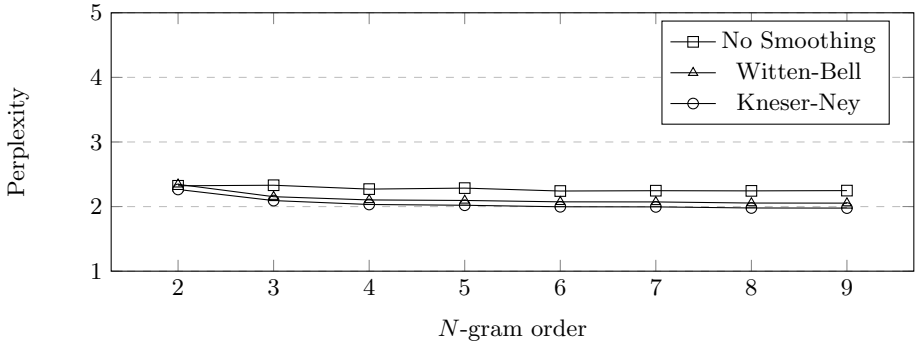


Fig. 3. Perplexity values for the JFreeChart API.

4.5 JSoup

The JSoup[6] library is used to fetch and parse HTML, manipulate and extract data from it, and clean it in order to prevent XSS attacks. It was expected that the JSoup API would have a relatively low perplexity in comparison with the APIs for creating graphical user interfaces.

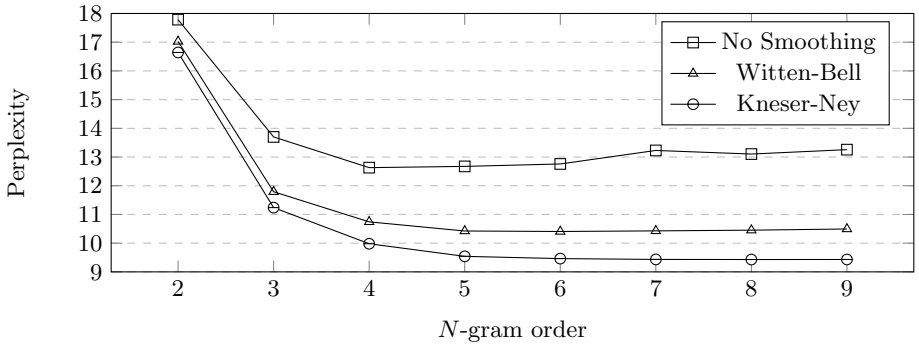


Fig. 4. Perplexity values for the JSoup API.

The high perplexity values (see Figure 4) have to do with the fact that it does not require a very standard way to be interacted with. We can also observe that without applying a smoothing technique, with $n \geq 5$, the perplexity values tend to increase. This is where the smoothing techniques take advantage and produce more accurate models, as we can see by the perplexity values achieved with the Witten-Bell and Kneser-Ney methods.

[6] http://jsoup.org/

4.6 JDBC Driver for MySQL

MySQL[7] provides drivers in several programming languages to make connections and execute statements in a database, enabling developers to integrate their applications with the MySQL databases.

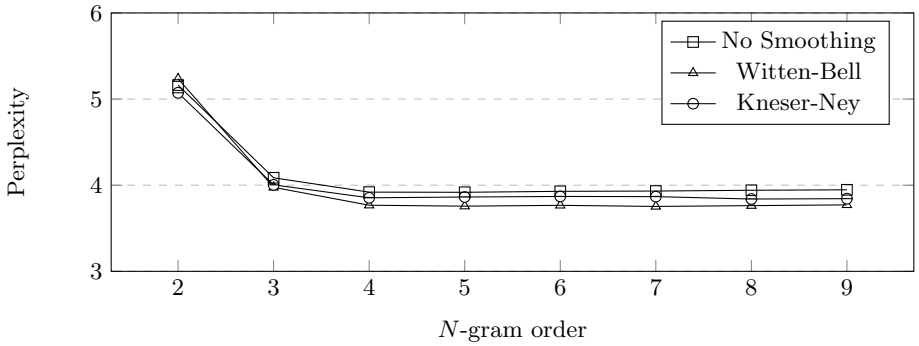


Fig. 5. Perplexity values for the JDBC Driver API.

Our intuition is that the low perplexity values (see Figure 5) are explained with the relatively simple usage of the API, since in most of the cases it basically requires to open a connection (with more or less properties), execute an SQL statement, and iterate over the results.

4.7 Jackson-core

This library[8] is used to process JSON data format. Its core contains a parser and an abstract generator used by its data processor.

Again, a low perplexity was expected due to its simple usage, that mainly requires a factory or a mapper, and a parser to read and a generator to write. In this API the smoothing methods did not improve accuracy significantly.

5 Discussion

We now compare and discuss the obtained results in order to draw some conclusions regarding the language models and their applicability in the domain of the APIs. Since in most of the cases the Kneser-Ney smoothing technique produced better perplexity values, Figure 7 compares those values across all the analyzed APIs.

[7] http://dev.mysql.com/downloads/connector/j/
[8] http://wiki.fasterxml.com/JacksonHome

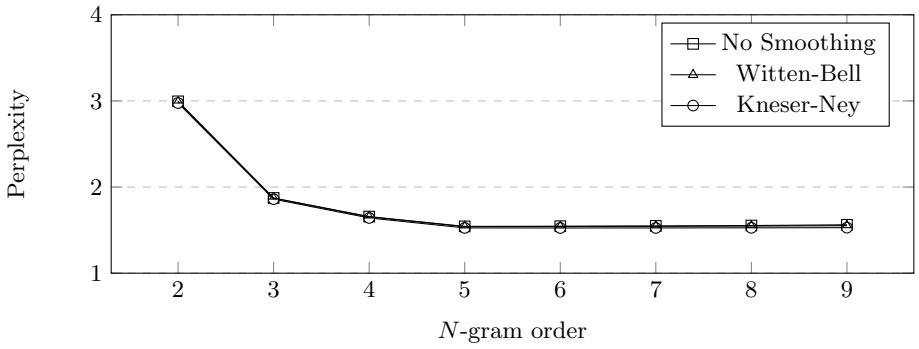


Fig. 6. Perplexity values for the Jackson-core API.

Regarding the perplexity, it is reasonable to conclude that APIs have a similar nature, i.e., as we increase the value of n, the perplexity decreases and stabilizes at n=6 in most cases. Even though these APIs may have very different purposes, this perplexity decrease was expected since as history increases in the n-gram models, there are less options for each history, but with more accurate estimates for that history.

Figure 8 presents the relation between the API's vocabulary size and the obtained perplexity values. Even though there are some outliers in this chart, the trendline suggests that as the size of the API's vocabulary increases, in general, the perplexity tends to increase as well. These results are not very surprising, because as the size of the API increases there may be more possible usage patterns. Nevertheless, this is not true in all the cases such as the JFreeChart API, that has a vocabulary similar to the SWT API and a lower perplexity. This has to do with the nature of the API, which has a relatively simple usage, and with the fact that the corpora may not cover a considerable part of the API.

N-gram language models achieve a perplexity that goes from 50 to 1000 on English text. Hindle et al. [11] obtained perplexity values that vary from approximately 3 to 32 for Java source code in general. These results are very encouraging because they show that API usage is far more regular than source code in general, which motivates our future work of creating a tool to help programmers find the next token they need to correctly interact with the API.

One of the main threats to the use of n-gram language models is the size of the training corpus, which may not contain all the possible sentences to populate the model. Although the smoothing techniques help mitigate this question, the model is not accurate when predicting sentences that did not occur in the training corpus. This is one of the most important factors to produce more accurate language models, and consequently better recommendations to the programmer.

It is a fact that the API's vocabulary is not as extensive as for example the English vocabulary. Even though the source code corpora might not include all the possible tokens, we argue that it contains enough to create models that

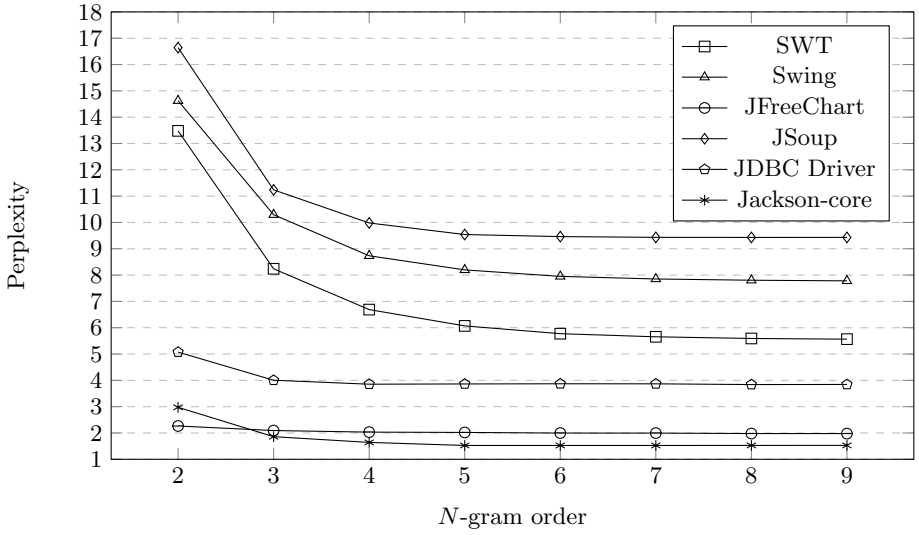


Fig. 7. Perplexity values comparison for all the APIs.

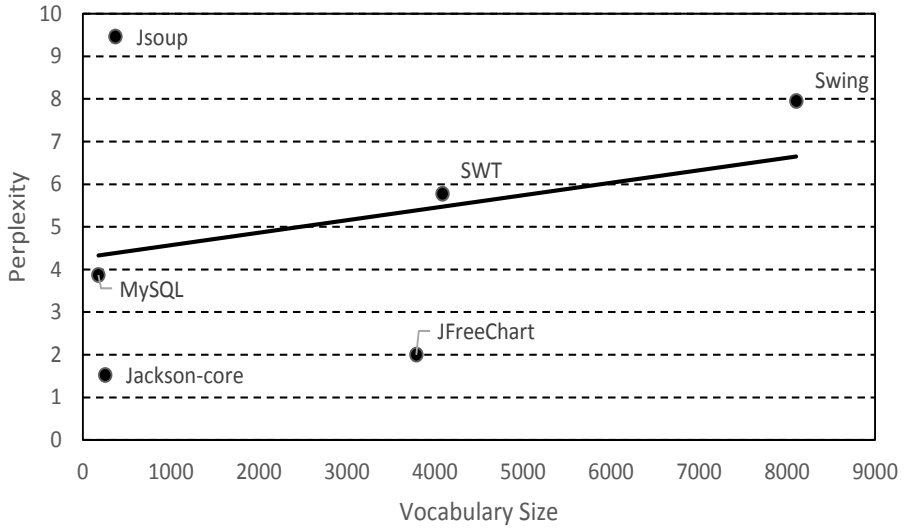


Fig. 8. Relation between vocabulary size and perplexity.

will help introduce the programmer to the API and still allow for some extra exploration.

Programmers often look for examples to learn an unknown API. These usually represent possible usage patterns and are often required to correctly interact with it [1]. When the resources are ill-formed, programmers might not correctly use the proper constructs of these patterns in their code, and even if the code compiles, and apparently does what it is supposed to, internally it might work incorrectly, thus being more likely to produce errors in the future. There is no viable way of manually filtering out these patterns from the source code corpora, but an adequate dimension of the corpora helps to mitigate this question. This point is related to the previous one in the sense that in order to allow for some exploration, the uncommon patterns must be taken into account. Just because they are not common, does not mean they are not correct. Smoothing techniques produce a more uniform probability distribution in language models, i.e. they will increase the probability of rare tokens and decrease the probability of the very common ones.

6 Related Work

Hindle et al. [11] describe that source code in general can be treated as a natural language, thus allowing to create language models to produce recommendation tools to help programmers. It was one of the papers that motivated our work on API usability and exploration using language models. Their approach differs from ours in the aspect that we focus only on creating models that can represent the API usage, which is much more restrict and regular than source code in general, thus allowing to produce more accurate recommendations.

Recently, a tool called SLANG [16] used n-gram language models and recurrent neural networks to fill holes in partial programs that use a certain API. They also replaced very rare API calls with an unknown token in order compact the language model. Since their work focuses on evaluating the tool and not the language models, we cannot compare the impact of this replacement. Even though our work only focuses on the language models, we intend to achieve a stepwise assistance tool based on a code completion system, that adapts and helps the programmer using and discovering the API and not by filling holes in partial programs.

SLAMC [13] is a statistical semantic language model for source code, where Nguyen et al. introduce semantic information into the language models, which represents additional information in the language model that would allow, for example, the global context of source files to help predict the next token. Tu et al. [20], however, argue that code tokenization is enough for n-gram language models. They also state that n-gram models will not help when a particular context is not present in the source code corpora used to train the model. On the one hand, we agree that these models cannot provide accurate recommendations when the code is not present in the training set. On the other hand, we argue that if they do not appear regularly, they are context specific and the probabilities

computed with the smoothing techniques should be enough to recommend these tokens.

Although it uses a different approach to produce the recommendations, it is important to mention the Code Recommenders [6] tool. This is an Eclipse Project that uses a modified k-nearest-neighbours [3] called "Best Matching Neighbours", that recommends method calls for objects in the context. It also provides features like "Override Completion" which recommends methods that should be overridden; "Chain Completion" returns a chain of method calls that return a desired type, which is useful when using for example object factories, that require invoking static methods to obtain the desired object; and, finally an "Adaptive Template Completion" that suggests a sequence of methods that usually occur together on a method.

7 Conclusions

In this paper we explored how language models perform on capturing regularities when using APIs, and which smoothing methods provide the best performance results. The obtained results show that language models can successfully be used to capture APIs usage patterns, with perplexity values ranging from 1.52 to 9.46 with the Kneser-Ney smoothing technique at $n = 6$, using interpolation.

There are some code completion systems that help the programmers explore and correctly use unknown APIs. However, these approaches can only recommend method calls for a certain object, or return the desired object type through a sequence of API calls. This requires the programmer to have some knowledge about the API.

As future work, we plan to develop a code completion tool to recommend the most likely tokens the programmer might need, according to the history (the code previously written). Additionally, user studies with programmers should be carried out in order to better assess the usability of such tool.

Regarding the language models, we intend to take into account the method's parameters in order to produce a more accurate model, since a token found in a block of code might depend on these variables.

References

1. Acharya, M., Xie, T., Pei, J., Xu, J.: Mining api patterns as partial orders from source code: from usage scenarios to specifications. In: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. pp. 25–34. ACM (2007)
2. Allamanis, M., Sutton, C.: Mining source code repositories at massive scale using language modeling. In: Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on. pp. 207–216. IEEE (2013)
3. Bobadilla, J., Ortega, F., Hernando, A., Gutiérrez, A.: Recommender systems survey. Knowledge-Based Systems 46, 109–132 (2013)

4. Brants, T., Popat, A.C., Xu, P., Och, F.J., Dean, J.: Large Language Models in Machine Translation. In: Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. pp. 858–867. Association for Computational Linguistics (2007)
5. Brown, P.F., deSouza, P.V., Mercer, R.L., Pietra, V.J.D., Lai, J.C.: Class-Based n-gram Models of Natural Language. Computational Linguistics 18(4), 467–479 (1992)
6. Bruch, M., Monperrus, M., Mezini, M.: Learning from examples to improve code completion systems. In: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering. pp. 213–222. ESEC/FSE '09 (2009)
7. Chen, S.F., Goodman, J.: An empirical study of smoothing techniques for language modeling. In: Proceedings of the 34th annual meeting on Association for Computational Linguistics. pp. 310–318. Association for Computational Linguistics (1996)
8. Chen, S.F., Goodman, J.: An empirical study of smoothing techniques for language modeling (1998)
9. Duala-Ekoko, E., Robillard, M.P.: Asking and answering questions about unfamiliar APIs: An exploratory study. In: Proceedings of the 34th International Conference on Software Engineering. pp. 266–276. ICSE '12 (2012)
10. Goodman, J.T.: A bit of progress in language modeling. Computer Speech & Language 15(4), 403–434 (2001)
11. Hindle, A., Barr, E.T., Su, Z., Gabel, M., Devanbu, P.: On the naturalness of software. In: Proceedings of the 34th International Conference on Software Engineering. pp. 837–847. ICSE '12, IEEE Press, Piscataway, NJ, USA (2012)
12. Kneser, R., Ney, H.: Improved backing-off for m-gram language modeling. In: Proceedings of 1995 International Conference on Acoustics, Speech, and Signal Processing. pp. 181–184. ICASSP'1995, IEEE (1995)
13. Nguyen, T.T., Nguyen, A.T., Nguyen, H.A., Nguyen, T.N.: A statistical semantic language model for source code. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. pp. 532–542. ACM (2013)
14. Piccioni, M., Furia, C.A., Meyer, B.: An empirical study of API usability. In: ESEM'13. pp. 5–14 (2013)
15. Pirinen, T.A., Hardwick, S.: Effect of Language and Error Models on Efficiency of Finite-State Spell-Checking and Correction. In: Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing. pp. 1–9. Association for Computational Linguistics (2012)
16. Raychev, V., Vechev, M., Yahav, E.: Code completion with statistical language models. In: Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. p. 44. ACM (2014)
17. Roark, B., Saraclar, M., Collins, M.: Discriminative n-gram language modeling. Computer Speech & Language 21(2), 373 – 392 (2007)
18. Robillard, M.P.: What makes APIs hard to learn? answers from developers. Software, IEEE 26(6), 27–34 (2009)
19. Stolcke, A., et al.: Srilm-an extensible language modeling toolkit. In: INTERSPEECH (2002)
20. Tu, Z., Su, Z., Devanbu, P.: On the localness of software. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 269–280. ACM (2014)
21. Zhong, H., Xie, T., Zhang, L., Pei, J., Mei, H.: MAPO: Mining and recommending API usage patterns. In: Proceedings of the 23rd European Conference on Object-Oriented Programming. pp. 318–343. ECOOP '09 (2009)

Deteção Automática de Plágio em Dois Atos

Bruno Felipe e João Cordeiro

m4622@ubi.pt, jpaulo@di.ubi.pt,
Universidade da Beira Interior, DI - Hultig,
Rua Marquês d'Ávila e Bolama, 6200 - 001 Covilhã, Portugal

Resumo O crescente aumento da quantidade de informação publicada na *Web*, na forma de publicações literárias, científicas e académicas, implica uma constante verificação da integridade de novos documentos (suspeitos) em função dos documentos existentes (fonte). Surge, portanto, a necessidade de aumentar: a eficiência na redução do espaço de procura em grandes conjuntos de documentos fonte; a eficácia na deteção de plágios cada vez mais sofisticados. Este artigo descreve uma metodologia baseada em dois atos: (i) Indexação do corpus fonte, com um motor de pesquisa (código aberto), e extração de documentos fonte (candidatos), através de pesquisa por palavras relevantes e caraterísticas textuais; (ii) localização de zonas de plágio em documentos suspeitos, com uma métrica híbrida, criada através da aplicação de programação genética sobre as caraterísticas de dados plagiados. Os resultados obtidos mostram uma redução significativa no tempo de processamento, devido à estratificação do corpus, assim como uma elevada taxa sucesso na deteção de zonas frásicas de plágio literal, modificado e ofuscado.

1 Introdução

No atual contexto da sociedade de informação, milhares de documentos emergem diariamente na Web, num espectro que começa com os simples comentários em blogs e redes sociais e termina com as elaboradas obras técnicas, científicas e literárias, passando necessariamente pelas inúmeras publicações jornalísticas, quer noticiosas, quer de comentário ou crónica.

A variedade e riqueza textual disponível ao utilizador comum têm crescido a ritmos exponenciais e de forma ímpar na história da humanidade. Esta nova realidade implica necessariamente o aumento de problemáticas e irregularidades clássicas, no domínio da publicação e autoria, sendo atualmente o plágio a mais preocupante. Por exemplo, no domínio da publicação académica e científica, esta é uma preocupação que está na ordem do dia com muitas situações detetadas de trabalhos plagiados, total ou parcialmente, havendo até algumas situações mediáticas recentes. Várias universidades já adotaram formalmente códigos deontológicos relativamente a esta matéria, com o intuito de *bem* formar os seus alunos. Todavia, as prevaricações continuam a acontecer.

Neste contínuo fluir de novos documentos, propelido pelas *tecnologias de informação e comunicação* (TIC), a tarefa de detetar plágio de forma manual

converge rapidamente para a impossibilidade absoluta ou pelo menos para o demasiado improvável. Para combater este *modernizado* antigo crime, torna-se fulcral fazer uso das mesmas TICs. Deste modo, a deteção automática de plágio é focada na análise de documentos suspeitos, procurando excertos de texto copiados de documentos fonte, sem o devido conhecimento e autorização do autor. Esta análise, designada por *deteção externa de plágio*, recorre a semelhanças léxico-gramaticais (lexical, sintática, semântica e estrutural) entre excertos frásicos do documento suspeito e uma coleção de documentos fonte (corpus fonte) [1]. Entretanto, o tamanho do corpus fonte pode variar de uma centena de documentos até à totalidade dos documentos contidos na *Web*.

Surge, portanto, a necessidade de reduzir o tamanho do corpus fonte para um conjunto menor de documentos (candidatos) com o mesmo género de informação do documento suspeito. O passo seguinte reside na análise frásica da informação partilhada entre os pares (documento suspeito versus documentos candidatos). Esta análise, detalhada e exaustiva, depende de uma métrica capaz de reconhecer as cópias ilícitas de informação. Surge, portanto, a necessidade de uma métrica robusta para detetar quer o plágio literal de palavras, frases e parágrafos, e etc, quer o plágio modificado com a reordenação das palavras mais relevantes, quer ainda o chamado *plágio ofuscado* que além da reordenação substitui palavras relevantes por seus sinónimos.

Este artigo aborda a eficiência na redução do corpus fonte para um pequeno conjunto de documentos candidatos, e a eficácia na deteção das zonas de plágio com uma métrica híbrida. Considerando que: se os atuais motores de pesquisa são eficientes na devolução das páginas *Web* pesquisadas, através de pesquisa com palavras-chave, então o mesmo pode ser aplicado para a procura de documentos candidatos, através extração de palavras informativas (*chave de pesquisa*) do documento suspeito e sua pesquisa no corpus fonte. Se os casos de plágio confirmado seguem padrões baseados no comportamento humano então será possível definir uma métrica, baseada nesses padrões, com o auxílio da inteligência computacional [2]. Este artigo está organizado da seguinte forma: a Secção 2 apresenta, sucintamente, uma revisão dos resultados e descobertas de outros estudos desta área; a Secção 3 descreve a metodologia desenvolvida; a Secção 4 descreve as experiências realizadas e os resultados obtidos; e a Secção 5 conclui o artigo.

2 Trabalho Relacionado

A deteção automática de plágio tem-se revelado uma tarefa desafiadora para a comunidade científica, na medida em que o universo documental, a partir do qual é possível plagiar, tem crescido de forma acelerada, nos últimos anos. Isto tem induzido algum trabalho recente nesta área [1,3,4]. Em termos concretos, sempre que colocamos em causa a honestidade de um trabalho escrito (*documento suspeito*), a confirmação da eventual fraude exige uma procura de dois tipos, etapas ou atos.

Num primeiro ato queremos identificar o subconjunto de documentos de onde possivelmente foram feitos os plágios, com o objetivo de reduzir o espaço de procura. Denominamos este subconjunto por *documentos fonte*. Esta designação é sempre relativa a um determinado documento suspeito. Podemos designar uma função Σ que mapeia um documento suspeito δ num subconjunto do universo[1] de documentos considerados $\mathcal{D}$, isto é: $\Sigma_{(\delta)} \subset \mathcal{D}$.

Num segundo ato, queremos localizar as possíveis passagens plagiadas do documento suspeito, para cada documento fonte. O primeiro ato é claramente um género de trabalho do âmbito da *Pesquisa de Informação*[2] [5,6]. O segundo ato envolve uma procura de similaridades entre fragmentos de textos (suspeito versus fonte), podendo ser ao nível do parágrafo, frásico ou sub-frásico. Este último ato parece mais simples, todavia podemos ter documentos longos e ainda em número elevado para comparar, tornando a tarefa morosa se não for bem realizada. Dado um documento suspeito δ e um conjunto candidato $\Sigma_{(\delta)} = \{d_1, ..., d_n\}$, queremos formas eficientes e eficazes de detetar se um par $\langle f_a(\delta), f_b(d_k)\rangle$ corresponde ou não a uma situação de plágio, sendo $f_a(d)$ o fragmento a de um documento d (ex: frase a desse documento).

Zechner et al. [7] propôs a redução do espaço de pesquisa documental com a aplicação de um modelo de espaço vetorial [8] para a indexação automática [9] do corpus ao nível frásico, conjugado com a medida de *similaridade do cosseno* [9] para a deteção de plágio entre pares frásicos. Em [10,11,12,13,14,15,16], a indexação do corpus fonte é efetuada com auxílio de motores de pesquisa[3] [4], com o código aberto, baseados neste mesmo modelo de espaço vetorial [17,18]. Para tal, a informação de cada documento fonte foi fragmentada em **subconjuntos** com tamanho fixo de palavras [10,12,13] ou frases [11,14,15,16] consecutivas, com sobreposição entre os subconjuntos indexados [10,12].

No âmbito do segundo ato são utilizadas diversas metodologias na análise detalhada entre os pares $\langle f_a(\delta), f_b(d_k)\rangle$. Em [10] foi utilizada a técnica *dotplot* [19] na análise detalhada, enquanto [11] utiliza sobreposições consecutivas de palavras (*word-n-gram*) [20,21]. Contrariamente, [12] utiliza a similaridade de *jaccard* [22,23] no segundo ato, ao mesmo tempo, [13] introduz e utiliza o *stripe-index* para a análise detalhada entre os pares. Em [14] foi criado um modelo de classificação, extraído do *algoritmo de classificação J48* [24]. Por outro lado, [15] utiliza o *fine-grained score* [25], a passo que [16] utiliza o algoritmo *running karp-rabin greedy string tiling* [26,27] na análise detalhada entre os pares.

As metodologias supracitadas foram apresentadas, por investigadores e profissionais oriundos da comunidade científica e industrial, tendo sido avaliadas [28] no âmbito da primeira, segunda e terceira *Competição Internacional de Deteção de Plágio* [29,30,31] organizadas pela PAN[5].

[1] Por exemplo, todos os documentos da *Web*.
[2] Information Retrieval.
[3] Apache Lucene™ - https://lucene.apache.org/
[4] Terrier - http://terrier.org/
[5] Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse: http://pan.webis.de/

3 Metodologia Proposta

Na abordagem que aqui propomos, qualquer documento d é visto como sendo constituído por uma sequência de m frases, $d = [s_1, s_2, ..., s_m]$, sendo, por sua vez, cada frase composta por uma sequência de palavras: $s_i = [w_1, w_2, ..., w_{n(i)}]$. De forma geral, representamos um *documento suspeito* de plágio pela letra delta: δ e um *documento fonte* por d, isto é, um documento a partir do qual poderá ter sido feito plágio. Em algumas abordagens considera-se uma granularidade ao nível do caracter, nós todavia considerámos a palavra como sendo a nossa unidade linguística mínima de análise. Tal como introduzido na Secção 2, dado um documento suspeito δ, podemos designar $\Sigma_{(\delta)} = \{d_1, ..., d_n\}$ o subconjunto *candidato* de todos os documentos fonte relativos a δ, de onde este obteve passagens plagiadas. Assim, um caso de plágio é um alinhamento entre dois fragmentos de texto, um de δ e outro de um $d \in \Sigma_{(\delta)}$, representado por: $\langle \delta^{\,a:A}, d^{\,b:B} \rangle$, sendo $a : A$ e $b : B$ excertos consecutivos dos respetivos documentos considerado. Por exemplo, o primeiro começa na frase a e terminando na frase A de δ. Sempre que se torne conveniente simplificar a notação e abstrair dos detalhes dos limites dos excertos, representaremos o último par por $\langle s_a, s_b \rangle$, sendo obviamente s_a um excerto de um δ e s_b um excerto de um d. A seguir são mostrados dois casos de plágios, obtidos de [28]:

```
Over the dog which is lazy jumps quickly the fox which is brown. (Sa)    (1)
The quick brown fox jumps over the lazy dog                      (Sb)

Dogs are lazy which is why brown foxes quickly jump over them.   (Sa)    (2)
The quick brown fox jumps over the lazy dog                      (Sb)
```

3.1 Primeiro Ato: Redução do Espaço de Pesquisa

No processo de deteção de plágio, em grandes coleções ($\mathcal{D}$) de documentos, a redução do espaço de procura tem-se revelado como um passo fundamental, prévio a uma análise mais minuciosa da procura dos excertos plagiados. Com este objetivo em mente, recorremos às potencialidades de indexação e pesquisa dos motores de pesquisa atuais. Um bom motor de pesquisa deve ser não só robusto mas também flexível, permitindo que suas bibliotecas de indexação e pesquisa sejam bem definidas, e possuam o código aberto. Este facto possibilita modificações e ajustes no código para finalidades específicas de indexação. Neste trabalho escolhemos o motor *Apache Lucene* [17] para indexar a nossa coleção $\mathcal{D}$, não esquecendo de fazer o devido pré-processamento e normalização dos documentos: remoção de elementos não textuais, como tabelas, gráficos, caracteres de pontuação e a redução do texto a minúsculas. Designe-se aqui a indexação desta coleção por *motor de plágio* (MP).

Após a indexação, estamos em condições de interrogar o MP, para um qualquer documento suspeito δ, com o objetivo de obter $\Sigma_{(\delta)}$. Há um desafio que aqui se coloca e que está relacionado com a informação de δ que deve ser selecionada e usada para interrogar o MP, uma vez que δ poderá ser longo e o MP aceita normalmente uma *chave de pesquisa* formada por um conjunto de palavras relevantes. É uma escolha que tem de ser feita e vários autores usam estratégias

diversas neste ponto. No nosso caso propomos aqui uma nova medida de seleção de palavras relevantes de um documento, para a formação da *chave de pesquisa* que será submetida ao MP. É uma medida que já temos vindo a experimentar desde [32] e que nos parece permitir de forma eficaz obter o grau de informação (importância) de qualquer palavra w num documento δ ou parte deste. É uma medida inspirada na medida TF*IDF de Salton [33], e que aqui designamos de *FIT – Função de Importância de Termo*, definida na Equação 1:

$$FIT(w) = |w| * \log_2 \left(\frac{P(w \mid \delta)}{P(w)} \right) \quad (1)$$

onde $|w|$ representa o comprimento de w, $P(w \mid \delta)$ a frequência relativa de w em δ e $P(w)$ a frequência relativa do mesmo termo w, mas estimado num corpus, por exemplo em $\mathcal{D}$. De uma maneira geral, quanto maior for uma palavra, mais a informação que veicula e logo maior será a sua importância. Assim, a função *FIT* permite-nos selecionar um conjunto de termos relevantes, representativos de um documento ou excerto deste. Esta noção de relevância é uma combinação do comprimento da palavra com o logaritmo de um rácio de ocorrências, no documento (δ) e num universo mais lato de documentos ($\mathcal{D}$).

Conjeturamos que em situações de plágio existirá necessariamente uma partilha de termos informativos, entre um documento suspeito e um documento fonte. Deste modo, selecionámos os μ termos mais relevantes de δ, com o valor $FIT(w) > 0$, para formarem a *chave de pesquisa* a submeter ao MP. O valor de μ poderá ser fixo a priori ou poderá depender de um limiar mínimo (β) de valor FIT. Podemos ainda escolher com base numa percentagem ($\alpha\%$) dos melhores termos. Estes critérios permitem a criação de uma *chave de pesquisa* consistente e adaptável para documentos extremamente grandes.

A submissão da *chave de pesquisa* ao MP devolverá um rank de documentos fonte $D = \{d_1, ..., d_n\}$. Deste conjunto, considerámos os ω primeiros documentos fonte como candidatos, deste modo, obtemos o conjunto $\Sigma_{(\delta)}$ a partir de $\mathcal{D}$. Com esta redução estamos assim em condições de passar ao segundo ato de deteção de passagens de plágio.

3.2 Segundo Ato: Deteção de Zonas de plágio

O segundo ato do nosso trabalho consistiu na procura e definição de uma métrica híbrida que reconheça os padrões de comportamento humano de plágio. Começámos por experimentar com cerca de duas dezenas de métricas [34] para o cálculo da similaridade entre excertos de texto, tais como a *BLEU*, *word-n-gram* overlap, similaridade Gaussiana, distância de Levenshtein, Sumo, entre outras. Após observarmos os valores obtidos, de cada métrica do conjunto, constatámos que os resultados estavam aquém do desejado. Consequentemente, desenhámos e experimentámos uma nova métrica, para o cálculo de similaridade entre excertos de texto à qual designamos de $nBinGram(.,.)$, apresentada na Equação 2, a qual nos parece mais adequada, tendo em conta certas especificidades léxicas

que habitualmente se verificam em pares de plágio.

$$nBinGram(x,y) = \sum_{k=1}^{n} \frac{\sqrt[3]{\frac{|x \wedge y|_k}{|x|_k} \times \frac{|x \wedge y|_k}{|y|_k} \times \frac{\min(|x|_k, |y|_k)}{\max(|x|_k, |y|_k)}}}{2^k} \quad (2)$$

Aqui, x e y representam dois excertos de texto, $|x|_k$ o número de sequências de k palavras que existem em x (igual para $|y|_k$), e $|x \wedge y|_k$ o número de sequências de k palavras comuns entre x e y. Deste modo, vemos que esta métrica também se baseia na contagem de sequências de palavras em comum, sendo uma combinação geométrica de três rácios, para cada k considerado. Cada parcela da soma vai também perdendo importância [35], uma vez que esta vai sendo dividida por 2^k.

Os resultados obtidos com a métrica $nBinGram(.,.)$ revelaram-se melhores, comparativamente a todas as outras métricas que experimentámos, para identificação de pares de plágio, mas ainda não satisfazendo completamente as nossas expectativas. Como resultado, resolvemos introduzir *aprendizagem automática* para induzir modelos caracterizadores dos padrões de plágio humano. Em particular, recorremos a algoritmos de *programação genética* com o objetivo de localizar as zonas de plágio em documentos suspeitos.

Utilizando um corpus de casos de plágio podemos explorar um conjunto de características potencialmente relevantes, assumindo que estas permitem, de alguma forma, modelar padrões de comportamento humano na execução de plágio. Designamos aqui este corpus por um conjunto de documentos suspeitos $D_{susp} = \{\delta_1, ..., \delta_m\}$, contendo documentos com excertos plagiados e outros "limpos", sem qualquer conteúdo plagiado. A estes últimos designamos de documentos *white* e aos primeiros de *black*. Isto é, podemos ter $v(\delta_i) \in \{black, white\}$, sendo $v(.)$ a função de *valoração* de um documento. Neste domínio e além de D_{susp}, existe um outro conjunto de documentos dos quais foram retirados os excertos para o plágio. Designamos este conjunto de $D_{font} = \{d_1, ..., d_n\}$. Um caso de plágio é uma referência que mapeia o início (a) e o fim (A) de um fragmento de texto de um documento $\delta \in D_{susp} \wedge v(\delta) = black$ no início (b) e fim (B) de um fragmento de texto de um documento $d \in D_{font}$, do género $\langle \delta^{\,a:A}, d^{\,b:B} \rangle$, tal como mencionado anteriormente. Normalmente, um documento *black* δ terá uma ou mais referências de plágio: $\texttt{refs}(\delta) = \{r_1, ..., r_p\}$, sendo $r_a = \langle \delta^{\,a:A}, d^{\,b:B} \rangle$, ou ainda de forma simplificada por $\langle s_a, s_b \rangle$.

Para uma referência $r_a = \langle s_a, s_b \rangle$ a relação de plágio entre s_a e s_b possui características próprias, inerentes ao grau de ofuscação perpetrado pelo infrator. Assim, neste trabalho considerámos um conjunto de características extraídas para cada caso r_a. Na Tabela 1 são apresentadas as principais características extraídas, onde "x" representa uma característica específica, por exemplo, para os **vocabulários** de ambos os excertos $V(s_a)$ e $V(s_b)$ e sua interceção $V(s_a) \cap V(s_b)$, denominado de vocabulário partilhado. Os **vocabulários únicos** de s_a e s_b, portanto $V(s_a) \backslash V(s_b) = voc_{unic}(s_a)$ e $V(s_b) \backslash V(s_a) = voc_{unic}(s_b)$. Considerámos também o grau de semelhança de sinónimos[6] entre estes dois últimos conjuntos,

[6] Usámos a WordNet: https://wordnet.princeton.edu/

designando-se este último atributo por $sino(.)$. Os **termos vulgares**[7] $sw(.)$ do $voc(.)$, do $voc_{part}(.)$ e do $voc_{unic}(.)$; os **os termos não vulgares** $nsw(.)$ de um vocabulário $voc(.)$, do $voc_{part}(.)$ e do $voc_{unic}(.)$; o uso de **sufixos**[8] $stem(.)$.

Tabela 1: Características extraídas de dados com plágio confirmado.

Técnica de Extração	Vocabulário				Vocabulário Único				Vocabulário Partilhado			
	$voc(x) = \{w_{x_1}, w_{x_2}, ..., w_{x_n}\}$				$voc_{unic}(x,y,z) = \lvert x \rvert - \lvert y \cap z \rvert$				$voc_{part}(x,y) = \lvert x \cap y \rvert$		$voc_{part}(x,y,z) = \lvert x \rvert \cap \lvert y \cap z \rvert$	
	$x = d_{black}$	$x = d_{font}$	$x = s_a$	$x = s_b$	$x = voc(d_{black})$ $y = voc(d_{black})$ $z = voc(d_{font})$	$x = voc(d_{font})$ $y = voc(d_{black})$ $z = voc(d_{font})$	$x = voc(s_a)$ $y = voc(s_a)$ $z = voc(s_b)$	$x = voc(s_b)$ $y = voc(s_a)$ $z = voc(s_b)$	$x = voc(d_{black})$ $y = voc(d_{font})$	$x = voc(s_a)$ $y = voc(s_b)$	$x = voc(s_a)$ $y = voc(d_{black})$ $z = voc(d_{font})$	$x = voc(s_b)$ $y = voc(d_{black})$ $z = voc(d_{font})$
$voc(.)$	x	x	x	x	x	x	x	x	x	x	x	x
$voc_{unic}(.)$	x	x	x	x	x	x	x	x	x	x	x	x
$voc_{part}(.)$	x	x	x	x	x	x	x	x	x	x	x	x
$sw(voc(.))$	x	x	x	x	x	x	x	x	x	x		
$sw(voc_{unic}(.))$	x	x	x	x	x	x	x	x	x	x		
$sw(voc_{part}(.))$	x	x	x	x	x	x	x	x	x	x		
$nsw(voc(.))$	x	x	x	x	x	x	x	x	x	x		
$nsw(voc_{unic}(.))$	x	x	x	x	x	x	x	x	x	x		
$nsw(voc_{part}(.))$	x	x	x	x	x	x	x	x	x	x		
$stem(voc(.))$			x	x			x	x				
$stem(voc_{unic}(.))$			x	x			x	x				
$stem(voc_{part}(.))$			x	x			x	x				
$stem(sw(voc(.)))$			x	x			x	x				
$stem(sw(voc_{unic}(.)))$			x	x			x	x				
$stem(sw(voc_{part}(.)))$			x	x			x	x				
$stem(nsw(voc(.)))$			x	x			x	x				
$stem(nsw(voc_{unic}(.)))$			x	x			x	x				
$stem(nsw(voc_{part}(.)))$			x	x			x	x				
$sino(.)$							x	x				

Após a coleta, de todas as características de todas as referências $\texttt{refs}(\delta) = \{r_1, ..., r_p\}$, de todos os $\delta \in D_{susp}$, são formadas as instâncias que serão utilizadas na aprendizagem automática. Estas foram guardados com o formato $ARFF$[9] definido pelo $Weka$[10]. Este formato especifica o armazenamento por instâncias e atributos de instâncias. Cada instância representará uma r_a de um $\texttt{refs}(\delta)$ e cada atributo de instâncias uma caraterística dessa referência

Na procura por padrões nos dados com plágio (instâncias e atributos de instâncias) tirámos partido das vantagens oferecidas pela programação genética [38]. A principal vantagem é constituída pela habilidade em descobrir *regularidades* em grandes quantidades de dados [39]. Por outras palavras, aprender os dados conhecidos e generalizar (programas representados com árvores) em dados desconhecidos [39]. A programação genética é considerada como uma especialização

[7] stopwords - palavras muito frequentes [36]
[8] stemming - extração de sufixos de palavras em Inglês [37]
[9] Attribute-Relation File Format
[10] Not the bird! Like the bird! - http://www.cs.waikato.ac.nz/ml/weka/

dos algoritmos genéticos [2,40], pelo facto de representar cada cromossoma como um programa executável [41,42,2]. Cada programa é composto por um conjunto terminal de variáveis e/ou constantes, e por um conjunto de funções matemáticas (exp, log, sin, cos, pow, $\sqrt{}$, min e/ou max [43]), funções aritméticas (+, −, / e/ou × [43]), funções booleanas ($\wedge$, $\vee$, $\oplus$ e/ou $\neg$ [43]) e/ou estruturas de decisão (*se-então-senão*, $>$ e $<$ [43]) [2], aplicáveis ao conjunto terminal [2,40]. Sendo representado sob a forma de uma árvore, com nodos (funções e/ou estruturas de decisão) e folhas (variáveis e constantes) [38,2], como no exemplo apresentado na Figura 1a.

Temos assim a geração de conhecimento simbólico (matemático) para caracterizar casos de plágio. No nosso problema, considerámos cada árvore como sendo um possível padrão de plágio (até então, desconhecido) composto por nodos (regras de comportamento) e folhas (características/atributos de instâncias). Logo, a partir das $r_a = \langle s_a, s_b \rangle$ (instâncias) e suas características (atributos de instâncias), o algoritmo de programação genética irá analisar as relações de plágio, e com base nas regras de comportamento entre as características, e gerar vários, possíveis, padrões (árvores) desconhecidos. Um padrão válido será aquele que melhor generalizar as relação de s_a com s_b sem perder a eficácia em detetar as zonas de plágio mais ofuscadas. O melhor padrão encontrado pela nossa metodologia pode ser observado na Figura 1b, sendo a função $Tab(.)$ definida pela Equação 3.

$$Tab(s_b) = \frac{\mid V(s_b) \cap V(\delta) \cap V(d) \mid}{\mid V(s_b) \mid} \tag{3}$$

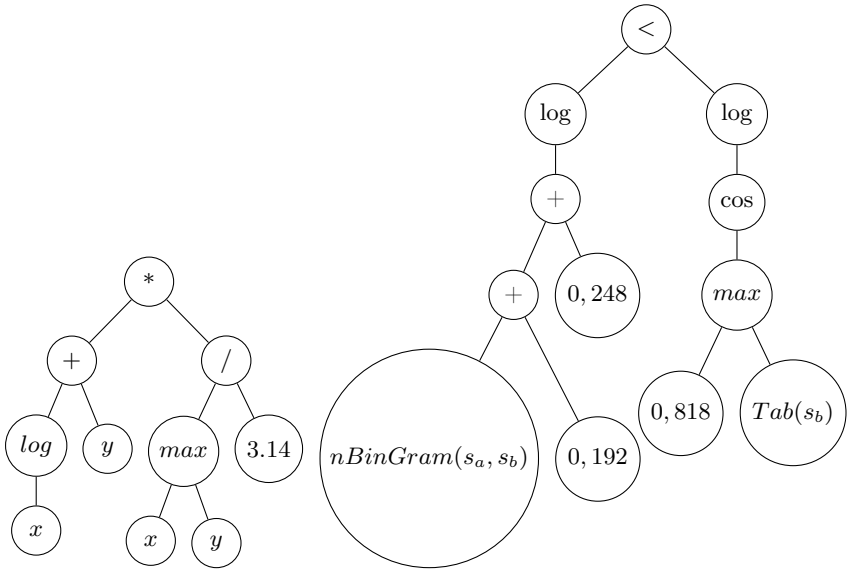
Com este padrão, denominado de *gp24n*, podemos definir a nossa métrica híbrida $MH(.,.)$, como sendo proveniente de várias características, conforme apresentado na Equação 4:

$$MH(s_a, s_b) = \begin{cases} 1 \text{ se } \log(0.45 + nBinGram(s_a, s_b)) > \log\Big(\cos\big(\max(0.818, Tab(s_b))\big)\Big) \\ 0 \text{ caso contrário} \end{cases} \tag{4}$$

onde $nBinGram(s_a, s_b)$ representa a métrica de similaridade proposta inicialmente, definida na Equação 2 e $Tab(s_b)$ representa uma função de extração de característica, apresentada na Tabela 1 e definida na Equação 3. Portanto, a deteção do plágio ocorrerá se e só se a condição imposta por $MH(.,.)$ for verdadeira (um) para os dois excertos da referência de plágio considerada.

4 Experimentação e Resultados

Nesta secção abordamos as experiências realizadas e os resultados obtidos para cada ato da metodologia descrita anteriormente. Primeiro, descrevemos o corpus, as ferramentas e os parâmetros utilizados para desenvolver e testar a nossa metodologia. No final, apresentamos os resultados de desempenho [44] obtidos para cada ato da nossa metodologia.



(a) Exemplo de uma árvore na programação genética.

(b) *gp24n*

Figura 1: Programas representados sob a forma de árvores.

Utilizámos o corpus *PAN-PC-11*[11] para desenvolver, testar e avaliar a nossa metodologia. Este corpus foi desenvolvido [28] e atualizado [31] para a avaliação de algoritmos de deteção automática de plágio em língua inglesa, sendo constituído por 11093 documentos fonte (o nosso D_{font}) e por 10538 documentos suspeitos (o nosso D_{susp}). O conjunto de documentos suspeitos é constituído por 5546 documentos sem plágio (D_{white}) e por 4992 documentos com plágio (D_{black})[12]. Conforme descrito anteriormente, na Secção 3.2, um δ_{black} possui uma ou mais referências de plágio, representadas por `refs`(δ_{black}). No corpus *PAN-PC-11*, estas estão descritas em ficheiros com extensão *xml*[13] com o mesmo nome do documento suspeito a que se referem.

O *Precision*, o *Recall* e o *F-Measure* foram as métricas utilizadas para avaliar os resultados experimentais obtidos de cada ato da nossa metodologia. Estas métricas, definidas originalmente em [44], para avaliar *sistemas de pesquisa de informação*[14], tornam-se *normas de facto*[15] na avaliação das classificações de algoritmos de aprendizagem automática [45], contrariamente às diferentes estruturas de avaliação [28,46], ao nível dos caracteres [29], propostos pela PAN nos

[11] http://www.uni-weimar.de/en/media/chairs/webis/corpora/pan-pc-11/
[12] $D_{susp} = D_{white} \cup D_{black}$
[13] Extensible Markup Language
[14] Information retrieval systems [44]
[15] De facto standard – https://en.wikipedia.org/wiki/De_facto_standard

últimos anos [29,47]. Nas Equações 5a, 5b e 5c, estão representadas as métricas de avaliação utilizadas, nomeadamente, o *Precision*, o *Recall* e o *F-Measure*, respetivamente.

$$Precision = \frac{TP}{TP + FP} \tag{5a}$$

$$Recall = \frac{TP}{TP + FN} \tag{5b}$$

$$F\text{-}Measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{5c}$$

Onde TP, FP, TN e FN, representam, respetivamente, os verdadeiros positivos, os falsos positivos, os verdadeiros negativos e os falsos negativos. Para visualizar-se melhor os resultados experimentais de cada ato da nossa metodologia, utilizámos a matriz de confusão. Logo, a Tabela 2, inspirada em [45], apresenta uma matriz de confusão "genérica" para as predições da classe realizadas por um "sistema genérico" em função do valor real da classe. Sendo: a soma dos TP com os FN igual aos reais elementos constituintes da classe 1, e a soma dos FP com os TN igual aos reais elementos constituintes da classe 0; a soma dos TP com os FP igual aos elementos preditos como da classe 1, com os TP como acertos e com os FP como erros, respetivamente, para essa classe 1; e, finalmente, a soma dos FN com os TN igual aos elementos preditos como da classe 0, tendo os TN como acertos e os FN como erros, respetivamente, para essa classe 0.

Tabela 2: Matriz de confusão para as predições da classe em função do valor real da classe. Inspirada em [45].

		Predição da classe		
		1	0	Total:
Classe	1	TP	FN	-
	0	FP	TN	-
Total de predições:		-	-	

Na redução do espaço de pesquisa, primeiro ato, utilizámos o motor de pesquisa *Apache Lucene*[16]. Escolhemos esta ferramenta pelo facto de esta possuir o código aberto e uma extensa e detalhada documentação associada [17]. Utilizámos as bibliotecas de indexação para indexar os 11093 documentos fonte (D_{font}). No passo seguinte, aplicámos a função $FIT(.)$, definida na Equação 1 e descrita na Secção 3.1, para extrair as palavras para a construção da *chave de pesquisa* de cada documento suspeito e, a seguir, submetemos às bibliotecas de pesquisa do *Lucene*, ou seja, ao MP.

Experimentámos várias combinações com diferentes valores para os parâmetros de configuração, nomeadamente, μ, $\alpha\%$, β e ω. A combinação que mais se destacou foi para valores de $\alpha = 5\%$, $\beta = 100$ e $\omega = 100$, sendo $\mu = \min(\alpha\%, \beta)$.

[16] Apache Lucene™ – https://lucene.apache.org/core/

Logo, para cada documento suspeito δ, construímos a *chave de pesquisa* com $\mu = \min(5\%, 100)$ termos relevantes com os maiores valores dados pela função FIT. Assim como, para cada *chave de pesquisa* submetida ao MP, considerámos apenas $\omega = 100$ primeiros documentos fonte, retornados pelo MP, como candidatos válidos.

Os resultados da avaliação do primeiro ato, para as *chaves de pesquisa* submetidas ao MP, podem ser observados na Tabela 3, assim como na Tabela 4 onde são apresentados os valores da matriz confusão das *chaves de pesquisa* criadas, para todos os 10538 documentos suspeitos (D_{black} e D_{white}). Salienta-se que os resultados da avaliação correspondem aos sucessos e insucessos, das *chaves de pesquisa* submetidas ao MP, em encontrar pelo menos um documento candidato igual à um dos documentos fonte descritos nas referências de plágio. Como é possível observar, na Tabela 3, as *chaves de pesquisa* criadas para os 4992 documentos com plágio, D_{black}, comprovam a eficiência da função $FIT(.)$ alinhada com os paramentos de configuração, ao passo que, as *chaves de pesquisa* criadas para os 5546 documentos sem plágio, D_{white}, comprovam a eficácia e corroboraram o primeiro ato da nossa metodologia.

Tabela 3: Avaliação das chaves de pesquisa submetidas ao *motor de plágio* em função da classe.

Precision	**Recall**	**F-Measure**	**Classe**
0.964	0.862	0.910	1
0.887	0.971	0.927	0

Tabela 4: Matriz confusão das *chaves de pesquisa* submetidas ao *motor de plágio* do Lucene.

		Lucene(*chaves de pesquisa*)		
		D_{black}	D_{white}	Total:
Classe	1	4304	688	4992
	0	160	5386	5546
Total de predições:		4464	6074	

No início do segundo ato extraímos as características de plágio utilizando todas as referências de plágio ($\texttt{refs}(\delta_i) = \{r_1, ..., r_p\}$) dos documentos suspeitos ($D_{susp} = \{\delta_1, ..., \delta_m\}$) com plágio confirmado ($v(\delta_i) = black$) para os fragmentos de texto s_a ($s_a = \delta_i^{a:A} \in r_a \in \texttt{refs}(\delta_i)$) e s_b ($s_b = d_j^{b:B} \in r_a, d_j \in D_{font}$) menores ou iguais aos 1000 caracteres ($|s_a| \leq 1000$ e $|s_b| \leq 1000$). De igual modo, utilizámos todos os documentos suspeitos ($D_{susp} = \{\delta_1, ..., \delta_m\}$) sem plágio ($v(\delta_i) = white$), para extrair, aleatoriamente ($v(\delta_{random}) = black$) , pseudo-referências ($p\text{-}refs(\delta_i) = \{p\text{-}r_1, ..., p\text{-}r_p\}$) de pseudo-plágio ($v(\delta_i) = white$) com

documentos fonte ($D_{font} = \{d_1, ..., d_n\}$), seguindo os mesmos limites para os fragmentos de texto s_a ($s_a = \delta_i^{a:A} \in p\text{-}refs(\delta_i), \delta_i \in D_{susp} \ \wedge \ v(\delta_i) = white$) e s_b ($s_b = \delta_{random}^{b:B} \in \texttt{refs}(\delta_{random}), \ \ \delta_{random} \in D_{susp} \wedge v(\delta_{random}) = black$) menores ou iguais aos 1000 caracteres ($|s_a| \leq 1000$ e $|s_b| \leq 1000$).

Para além da classe, zero (0) e um (1), respetivamente, sem plágio e com plágio; foram extraídas 232 características (atributos de instâncias) para: 14746 referências de plágio (instâncias); e 12904 pseudo-referências (instâncias) de pseudo-plágio com os documentos fonte.

Tabela 5: Avaliação das deteções de zonas de plágio com a métrica híbrida $MH(.,.)$ em função da classe.

Precision	**Recall**	**F-Measure**	**Classe**
0.998	0.997	0.997	1
0.987	0.989	0.988	0

Tabela 6: Matriz de confusão para as deteções de zonas de plágio com a métrica híbrida $MH(.,.)$.

		MH(.,.)		
		1	0	Total:
Classe	1	2086	6	2092
	0	5	468	473
Total de predições:		2091	474	

Para procurarmos por padrões nos dados plagiados utilizámos a ferramenta[17], com o código aberto, desenvolvida por Yan Levasseur. Esta ferramenta, originalmente desenvolvida para o reconhecimento de objetos biológicos em imagens bidimensionais [43], constitui uma implementação eficaz do algoritmo de programação genética. A sua utilização está associada ao *software* de mineração de dados Weka como uma espécie de extensão, permitindo, utilizar o Weka como ligação entre a aplicação da ferramenta de programação genética e as características de plágio, armazenadas com formato *ARFF*[18] do Weka.

Extraímos 184 padrões de plágio a partir das características de plágio em combinação com a variação de diferentes parâmetros da ferramenta[19] de programação genética [43]. Estes parâmetros (e respetivas variações) são: o número de gerações (20, 100, 200, 1000 ou 10000); o tamanho da população (25, 50, 100 ou 1000); os tipos de reprodução (*cross-over*, mutação, mutação de nodos funcionais, reprodução e/ou novo programa [43]); os tipos de seleção (torneio ou

[17] http://www.leyan.org/tiki-index.php?page=Genetic%20Programming
[18] Attribute-Relation File Format
[19] http://www.leyan.org/tiki-index.php?page=Genetic%20Programming

proporcional ao valor de aptidão[20]); os operadores funcionais (+, −, /, ×, exp, log, sin, cos, pow, $\sqrt{}$, min, max, $\wedge$, $\vee$, $\oplus$, $\neg$, *se-então-senão*, $>$ e/ou $<$ [43]); os avaliadores de aptidão (soma de erros - contínuo [43], raiz quadrada do erro quadrático médio - contínuo [43]; ou confiança no reconhecimento da classe - classificador híbrido com impulsão[21] [48,43]); entre outros fatores. Como resultado, apresentamos aqui o padrão de plágio que mais se destacou, designado por *gp24n*, representado sob a forma de árvore na Figura 1b e sob a forma de equação na Equação 4. Aproveitamos ainda para salientar a elevada deteção de zonas de plágio, conseguida com esta equação, conforme apresentado nas Tabelas 5 e 6.

5 Conclusões

Neste artigo apresentamos uma metodologia eficiente e eficaz para deteção automática de plágio em dois atos. É certo que não comparámos diretamente com outros sistemas, participantes na competição *PAN-PC-11*[22], pois estes usaram o conjunto de dados só para teste, enquanto nós para treino, teste e validação. Por outro lado, alguns parâmetros de avaliação são um pouco diferentes, como por exemplo a sobreposição ao nível do caracter, entre outros (ver Secção 4). Portanto, a nossa afirmação de eficácia e eficiência baseiam-se aqui unicamente nos valores elevados de *precision* e *recall* obtidos.

Primeiro, mostramos que se os atuais motores de pesquisa são eficientes nas pesquisas com palavras-chave, então o mesmo pode ser aplicado para a procura de plágio. Propusemos uma técnica baseada na seleção de palavras informativas do documento suspeito de plágio (δ). Estas são depois integradas na formação da *chave de pesquisa* que será submetida ao *motor de plágio*. Desta forma, reduzimos significativamente o universo de documentos originais sem perder o subconjunto significativo ($\Sigma_{(\delta)}$), no qual se encontram os excertos plagiados em δ. Destacamos a obtenção de valores de **F-Measure** acima de 0.9 (Tabela 3).

Relativamente ao segundo ato, mostramos que os casos de plágio confirmados seguem padrões, que são baseados no comportamento humano e descrevemos como identificá-los de forma automática com o auxílio da inteligência computacional [2], em particular com o recurso à programação genética. A partir de um corpus de plágio (*PAN-PC-11*), extraímos um vasto conjunto de características, gerando instâncias que depois foram utilizadas para induzir os padrões de caracterização de pares de plágio. Além de gerarmos conhecimento simbólico (inteligível por humanos), obtivemos resultados muito animadores, com valores de **F-Measure** na ordem 0.98 (Tabela 5).

[20] Fitness
[21] Boosting
[22] http://www.uni-weimar.de/en/media/chairs/webis/corpora/pan-pc-11/

Referências

1. Alzahrani, S., Salim, N., Abraham, A.: Understanding plagiarism linguistic patterns, textual features, and detection methods. IEEE Trans. Syst., Man, Cybern., Syst., Part C **42**(2) (2012) 133–149
2. Engelbrecht, A.P.: Computational Intelligence: An Introduction. 2nd edn. Wiley Publishing (2007)
3. Maurer, H.A., Kappe, F., Zaka, B.: Plagiarism - a survey. J. Univers. Comput. Sci. **12**(8) (2006) 1050–1084
4. Stein, B., Koppel, M., Stamatatos, E.: Plagiarism analysis, authorship identification, and near-duplicate detection. SIGIR Forum **41**(2) (2007) 68–71
5. Canfora, G., Cerulo, L.: A taxonomy of information retrieval models and tools. J. Comput. Inf. Technol. **12**(3) (2004) 175–194
6. Singhal, A.: Modern information retrieval: A brief overview. IEEE Data Eng. Bull. **24**(4) (2001) 35–43
7. Zechner, M., Muhr, M., Kern, R., Granitzer, M.: External and intrinsic plagiarism detection using vector space models. In Stein, B., Rosso, P., Stamatatos, E., Koppel, M., Agirre, E., eds.: SEPLN 09 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse, CEUR-WS.org (2009) 47–55
8. Manning, C.D., Raghavan, P., Schütze, H.: Scoring, term weighting and the vector space model. In: Introduction to Information Retrieval, Cambridge, UK, Cambridge University Press (2009) 109–133
9. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Commun. of The ACM **18**(11) (1975) 613–620
10. Costa-Jussà, M.R., Banchs, R.E., Grivolla, J., Codina, J.: Plagiarism detection using information retrieval and similarity measures based on image processing techniques: Lab report for pan at clef 2010. In Braschler, M., Harman, D., Pianta, E., eds.: Notebook Papers of CLEF 2010 LABs and Workshops, 22-23 September, Padua, Italy. (2010)
11. Vania, C., Adriani, M., Komputer, F.I., Depok, K.: Automatic external plagiarism detection using passage similarities: Lab report for pan at clef 2010. In Braschler, M., Harman, D., Pianta, E., eds.: Notebook Papers of CLEF 2010 LABs and Workshops, 22-23 September, Padua, Italy. (2010)
12. Muhr, M., Kern, R., Zechner, M., Granitzer, M.: External and intrinsic plagiarism detection using a cross-lingual retrieval and segmentation system: Lab report for pan at clef 2010. In Braschler, M., Harman, D., Pianta, E., eds.: Notebook Papers of CLEF 2010 LABs and Workshops, 22-23 September, Padua, Italy. (2010)
13. Gottron, T.: External plagiarism detection based on standard ir technology and fast recognition of common subsequences: Lab report for pan at clef 2010. In Braschler, M., Harman, D., Pianta, E., eds.: Notebook Papers of CLEF 2010 LABs and Workshops, 22-23 September, Padua, Italy. (2010)
14. Pereira, R.C., Moreira, V.P., Galante, R.: Ufrgs@pan2010: Detecting external plagiarism: Lab report for pan at clef 2010. In Braschler, M., Harman, D., Pianta, E., eds.: Notebook Papers of CLEF 2010 LABs and Workshops, 22-23 September, Padua, Italy. (2010)
15. Ghosh, A., Bhaskar, P., Pal, S., Bandyopadhyay, S.: Rule based plagiarism detection using information retrieval: Notebook for pan at clef 2011. In: In Notebook Papers of CLEF 2011 LABs and Workshops, 19-22 September, Amsterdam, The Netherlands. (2011)

16. Nawab, R.M.A., Stevenson, M., Clough, P.D.: External plagiarism detection using information retrieval and sequence alignment: Notebook for pan at clef 2011. In: In Notebook Papers of CLEF 2011 LABs and Workshops, 19-22 September, Amsterdam, The Netherlands. (2011)
17. McCandless, M., Hatcher, E., Gospodnetic, O.: Lucene in Action, Second Edition: Covers Apache Lucene 3.0. Manning Publications Co., Greenwich, CT, USA (2010)
18. Ounis, I., Amati, G., V., P., He, B., Macdonald, C., Johnson: Terrier information retrieval platform. In: Proceedings of the 27th European Conference on IR Research (ECIR 2005). Volume 3408., Springer (2005) 517–519
19. Church, K.W., Helfman, J.I.: Dotplot: A program for exploring self-similarity in millions of lines for text and code. J. Comput. Graph. Stat. **2**(2) (1993) 153–174
20. Lyon, C., Malcolm, J., Dickerson, B.: Detecting short passages of similar text in large document collections. In: Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing. (2001) 118–125
21. Broder, A.: On the resemblance and containment of documents. In: Proceedings of the Compression and Complexity of Sequences 1997. SEQUENCES '97, Washington, DC, USA, IEEE Computer Society (1997) 21–29
22. Jaccard, P.: Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines. B. Soc. Vaud. Sci. N. **37** (1901) 241–272
23. Tanimoto, T.: An elementary mathematical theory of classification and prediction. IBM Internal Report (1958)
24. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)
25. Keselj, V., Peng, F., Cercone, N., Thomas, C.: N-gram-based author profiles for authorship attribution (2003)
26. Nawab, R.M.A., Stevenson, M., Clough, P.: University of sheffield: Lab report for pan at clef 2010. In Braschler, M., Harman, D., Pianta, E., eds.: Notebook Papers of CLEF 2010 LABs and Workshops, 22-23 September, Padua, Italy. (2010)
27. Wise, M.J., of Sydney, U.: Running Karp-Rabin matching and greedy string tiling. Basser Dept. of Computer Science, University of Sydney, Sydney (1993)
28. Potthast, M., Stein, B., Barrón-Cedeño, A., Rosso, P.: An evaluation framework for plagiarism detection. In Huang, C.R., Jurafsky, D., eds.: 23rd International Conference on Computational Linguistics (COLING 10), Stroudsburg, Pennsylvania, Association for Computational Linguistics (2010) 997–1005
29. Potthast, M., Stein, B., Eiselt, A., Barrón-Cedeño, A., Rosso, P.: Overview of the 1st international competition on plagiarism detection. In Stein, B., Rosso, P., Stamatatos, E., Koppel, M., Agirre, E., eds.: SEPLN 09 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse, CEUR-WS.org (2009) 1–9
30. Potthast, M., Barrón-Cedeño, A., Eiselt, A., Stein, B., Rosso, P.: Overview of the 2nd international competition on plagiarism detection. In Braschler, M., Harman, D., Pianta, E., eds.: Working Notes Papers of the CLEF 2010 Evaluation Labs. (2010)
31. Potthast, M., Eiselt, A., Barrón-Cedeño, A., Stein, B., Rosso, P.: Overview of the 3rd international competition on plagiarism detection. In Petras, V., Forner, P., Clough, P., eds.: Working Notes Papers of the CLEF 2011 Evaluation Labs. (2011)
32. Mendes, H.d.C.: Similaridade documental e detecção de plágio. In: Dissertação para obtenção do Grau de Mestre, Departamento de Informática, Universidade da Beira Interior, Covilhã (2013) 25–28
33. Salton, G., Buckley, C.: Term-weighting approaches in automatic text retrieval. Infor. Proces. Manage. **24**(5) (1988) 513–523

34. Cordeiro, J., Dias, G., Brazdil, P.: New functions for unsupervised asymmetrical paraphrase detection. J. Softw. **2**(4) (2007) 12–23
35. Barrón-Cedeño, A., Rosso, P.: On automatic plagiarism detection based on n-grams comparison. In Boughanem, M., Berrut, C., Moth, J., Soule'-Dupuy, C., eds.: Proceedings of the 31th European Conference on IR Research on Advances in Information Retrieval, Berlin, Heidelberg, Springer-Verlag (2009) 696–700
36. Stamatatos, E.: Plagiarism detection based on structural information. In: Proceedings of the 20th ACM Conference on Information and Knowledge Management. (2011) 1221–1230
37. Porter, M.: An algorithm for suffix stripping. Program-Electron. Lib. **14**(3) (1980) 130–137
38. Eggermont, J.: Data Mining using Genetic Programming: Classification and Symbolic Regression. PhD thesis, Institute for Programming research and Algorithmics, Leiden Institute of Advanced Computer Science, Faculty of Mathematics & Natural Sciences, Leiden University, The Netherlands (2005)
39. Brameier, M.F., Banzhaf, W.: Linear Genetic Programming. Springer US (2007)
40. Spector, L.: A genetic programming approach. In: Automatic Quantum Computer Programming. Volume 7 of Genetic Programming., Springer US (2004) 43–54
41. Koza, J.R.: Hierarchical genetic algorithms operating on populations of computer programs. In: Proceedings of the 11th International Joint Conference on Artificial Intelligence. Volume 1 of IJCAI'89., San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1989) 768–774
42. Koza, J.R.: Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems (1990)
43. Levasseur, Y.: Techniques de l'intelligence artificielle pour la classification d'objets biologiques dans des images bidimensionnelles. In: Mémoire de Maîtrise Électronique, École de Technologie Supérieure, Montréal. (2008) 50–90
44. Rijsbergen, C.J.V.: Information Retrieval. 2nd edn. Butterworth-Heinemann, Newton, MA, USA (1979)
45. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, Second Edition. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005)
46. Potthast, M., Hagen, M., Völske, M., Stein, B.: Crowdsourcing interaction logs to understand text reuse from the web. In Fung, P., Poesio, M., eds.: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics (2013) 1212–1221
47. Potthast, M., Gollub, T., Hagen, M., Graßegger, J., Kiesel, J., Michel, M., Oberländer, A., Tippmann, M., Barrón-Cedeño, A., Gupta, P., Rosso, P., Stein, B.: Overview of the 4th international competition on plagiarism detection. In Forner, P., Karlgren, J., Womser-Hacker, C., eds.: Working Notes Papers of the CLEF 2012 Evaluation Labs. (2012)
48. Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In Saitta, L., ed.: Proceedings of the Thirteenth International Conference on Machine Learning, Morgan Kaufmann (1996) 148–156

Fault-Tolerant Precision Time Protocol for Smart Grids

Radu Onica, Nuno Neves, Antonio Casimiro

LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal
ronica@lasige.di.fc.ul.pt, nuno@di.fc.ul.pt, casim@ciencias.ulisboa.pt

Abstract. The Precise Time Protocol defined in the IEEE 1588 standard is widely used to synchronize clocks, with a high degree of precision. The current needs of time synchronization in Smart Grids are satisfied with multiple GPS based clocks located around the network. This solution, however, is not cost efficient and if any of these GPS clocks is successfully attacked, a part of the network can be rendered unusable. This paper investigates a way of adding fault tolerance capabilities to the PTP protocol to address attacks aiming to interfere with the GPS based clock service. It is shown that the solution manages to detect anomalous clock behaviors and recover from them. Furthermore, the solution is able to deal with multiple GPS clock failures, as long as some correct time source is still available.

Keywords: Precision Time Protocol, GPS Clocks, Fault Tolerant

1 Introduction

Smart grids are composed of different types of components, each with its own requirements. Time synchronization plays a crucial role in the communication and proper functioning of several of these components. One component in particular, the Phase Measurement Unit (PMU), depends almost entirely on high accuracy time synchronization. The PMU measures and keeps track of various parameters (e.g., electrical current amplitude and phase) of the part of the smart grid where it is installed. This can be thought of as a "snapshot" of the state in which that portion of the smart grid is in. The deployment of multiple PMUs at different points across the smart grid allows for a complete state estimation. This information can then be used to support decisions on the need to modify the power generation or the distribution of loads, which in turn gives the ability to predict and avoid failures like blackouts. Time synchronization among all PMUs is critical to create a coherent picture of the smart grid. If PMUs take snapshots at different times, the complete state cannot be correctly estimated. Current PMUs use GPS receivers for time synchronization. This is a very costly solution because it requires dedicated hardware to be present in each and every PMU. From a security standpoint, it also leaves the PMUs, and in turn the whole smart grid, vulnerable to even the most basic GPS attacks like blocking or jamming [1].

1.1 Requirements

The requirements of a time synchronization protocol suitable for use in smart grids are explained in detail in [2]. The standard is split into two parts: the first one describes the measurement (called phasor in this context) estimation requirements while the second one explains the network communication protocol. The time synchronization requirements are presented as the maximum allowed error for PMUs. This is translated into a need for the PMUs to make their measurements at approximately the same time, within an interval of at most $\approx 30\mu$s.

Even though the standard does not talk about security requirements, the protocol needs to be relatively secure and maintain timing errors below the thresholds even in the case of an attack. The fact that each PMU is equipped with a GPS receiver means that it also inherits all the insecurity and problems related to GPS, which in turn makes the implementation of a secure time synchronization algorithm more difficult.

1.2 Precision Time Protocol

The precision time protocol appeared as a result of the growing need for high precision time keeping. It was first described in the IEEE 1588-2002 standard, officially entitled IEEE Standard for a Precision Clock Synchronization Protocol for Network Measurement and Control Systems. A revised version appeared in 2008 as the IEEE 1588-2008 standard [3]. It was designed to fill in the gap left open by the Network Time Protocol (NTP) and the Global Positioning System (GPS) time service. It offers far better accuracy than NTP, but without the need for a dedicated GPS receiver at each network node. GPS receivers can be used in combination with a PTP network by acting as a Grandmaster Clock, i.e., the time source for that network. A simple PTP network can be seen in [Figure 1]. It is composed of a Grandmaster Clock that acts as a time source for the network; a Boundary Clock that synchronizes itself to this Grandmaster Clock; the Boundary Clock further synchronizes another part of the network, namely a Slave Clock.

Since a PMU uses a GPS receiver to synchronize its local clock, it can be seen as being made up of a Grandmaster Clock directly connected to a Boundary Clock [Figure 1].

PTP uses the master-slave architecture for time distribution in which one or more clocks are used as well as various communication media (e.g., network links). It defines different roles for every clock in the network and different states for every communication port in use:

- **Grandmaster Clock:** is a clock that synchronizes directly to a GPS receiver. It always runs in PTP Master mode, meaning it will distribute its time throughout the network;
- **Ordinary Clock:** is a normal clock that synchronizes itself to another source. It always runs in the PTP Slave mode;

– **Boundary Clock:** is a clock that both synchronizes itself to a PTP Master (acting as a PTP Slave) and further distributes the time to another part of the network (acting as a PTP Master). **Only the PTP Slave port of the Boundary Clock has the ability to make changes to the internal clock**. The other ports can only read the local clock;
– **Transparent Clock**: is a special type of clock that modifies all PTP messages that go through this device, correcting the message timestamps to eliminate the time spent traversing the network equipment. This scheme improves the time distribution accuracy by compensating for the time that messages spend in each communication device (e.g., a switch).

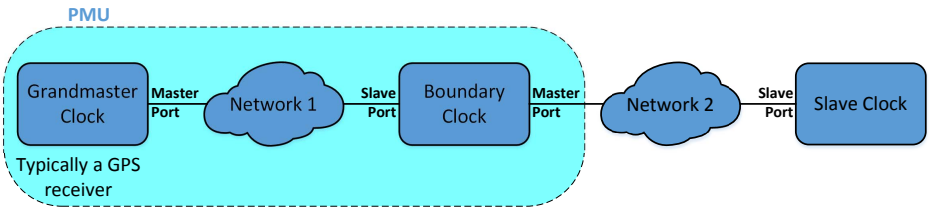


Figure 1: A simple PTP Network where a PMU uses a GPS receiver (displayed as a Grandmaster Clock) to synchronize its clock (represented as a Boundary Clock).

The synchronization is achieved by exchanging PTP messages between the master port of a clock and the slave port of another clock. The messages are divided into event messages and general messages. Event messages are timestamped with both transmission and reception times. General messages do not require accurate timestamps and are used for both synchronization and configuration purposes.

Based on timestamps, the slave calculates the time offset from the master clock and eventually adjusts its local time to be similar to the master. The basic synchronization message exchange is showed in [Figure 2] and the logic behind it is explained below :

1. The master sends a *Sync Message* to the slave and saves the time *t1* at which it was transmitted;
2. The slave receives the *Sync Message* and notes the time of reception *t2*;
3. The master conveys to the slave the timestamp *t1* by embedding it in a *Follow_Up Message*. Alternatively this timestamp can be embedded into the *Sync Message* using some kind of hardware processing (this is called the one-step mechanism);
4. The slave sends a *Delay_Req Message* to the master and saves the time *t3* at which it was transmitted;

5. The master receives the *Delay_Req Message* and notes the time of reception *t4*;
6. The master conveys to the slave the timestamp *t4* by embedding it in a *Delay_Resp Message*.

Considering that the clock offset of the slave relative to the master is Δ and that the network delay between the master and slave is *d(MS)* and between the slave and master is *d(SM)* we have:

$$t2 = t1 + \Delta + d(MS) \tag{1}$$

$$t4 = t3 - \Delta + d(SM) \tag{2}$$

Notice that we have two equations and three unknowns, which creates an impossibility of resolution. The PTP standard makes the assumption that the network delays are the same in both ways, i.e., *d(MS)=d(SM)=d*. Because of this (1) and (2) can be simplified:

$$d = \frac{t2 - t1 + t4 - t3}{2}$$

$$\Delta = t2 - t1 - d$$

It is now possible to calculate both the one way delay as well as the clock offset Δ. However, because of the aforementioned assumption, the calculated clock offset will have an error that is equal to the difference between the mean path delay and the master to slave delay.

The PTP standard utilizes an algorithm called *Best Master Clock algorithm* (BMC) to select among the clocks the one to serve as the master. This clock will provide reference time for the network. Selecting the best master takes into account various parameters of each node:

- **Clock quality** - is the expected deviation from real time. In version 2 of PTP, the clock quality is defined with two data fields, the *clockAccuracy* and *clockClass*. The *clockClass* is used for transitioning a port from one state into another in our implementation. By default, the values 13, 255 and 248 are used respectively to designate a master only port, a slave only port or a master/slave port. The Master/Slave port can run in either of these states. In a PTP network, one will be Master and the rest will be Slaves. The BMC algorithm is used to decide which one should be the Master. If a group of clocks is run in Master only mode, after the best one is selected, all others will go into a passive state (called PTP Passive);
- **Priority** - is represented by two 8-bit fields known as *Priority1* and *Priority2*;
- **Variance** - is an estimation of the stability of the clock based on performance observations;
- **Identifier** - is an universally unique numeric identifier of the clock and is used as a tie breaker if all the other parameters are the same.

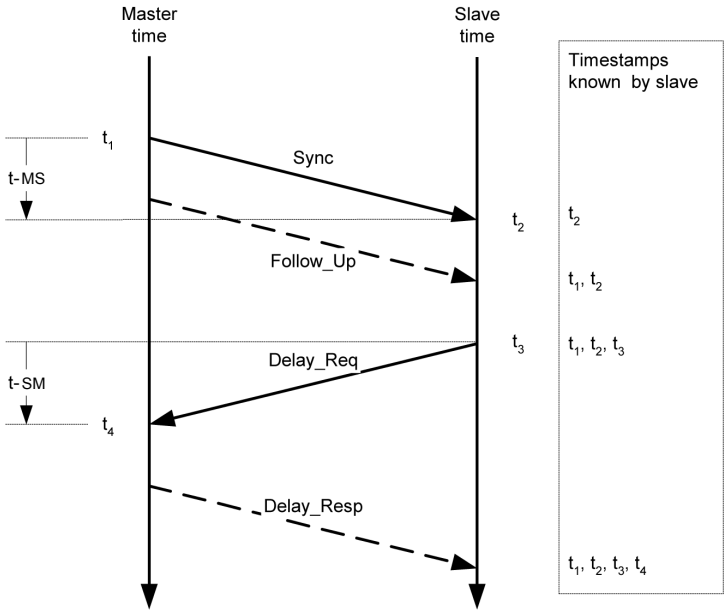


Figure 2: PTP Message Exchange.

1.3 Related work on PTP security

Several security related problems were discovered with PTP after the first version of the standard was released [4]. Annex K of the Precision Time Protocol standard **version 2** IEEE 1588-2008 [3] presents several security guidelines that can help to withstand different types of attacks. The guidelines are however completely optional and in an experimental state. They focus on adding two main security mechanisms:

- An integrity protection mechanism, which uses a Message Authentication Code (MAC) to verify that messages have not suffered any unauthorized modification in transit. A message counter is implemented to help prevent replay attacks;
- An authentication method based on a challenge-response mechanism.

A flag in the PTP message is used to indicate that it is carrying security related information. Each message following this one is required to also present the same flag or else it is silently discarded without wasting any more resources. In order to ease this process for hardware implementations, the flag should be the last field in the message. By doing this, an indirect protection against Denial of Service attacks on system resources is created.

In [5] a comprehensive threat analysis of the PTP protocol is done and solutions using IPsec and MACsec are explored.

GPS receivers are often used as the time source for the smart grid, which makes it a prime target for attackers. Some research has been previously done on various types of attacks on GPS receivers, like Blocking, Jamming [1] or Spoofing [6]. Blocking attacks try to isolate the receiver from the satellite signal leading to a signal loss. In jamming attacks, significant RF (radio-frequency) noise is transmitted so that the receiver can no longer get the satellite signal. Spoofing attacks generate fake GPS signals that cause the receiver to calculate an incorrect location and/or time solution.

Recently a more advanced type of attack that targets the receiver itself was demonstrated [7]. It alters the satellite signal data in such a way as to make the receiver output incorrect results or downright crash. These types of attacks are becoming more common because of the relatively cheap cost of the Spoofer Device (called Phase-Coherent Signal Synthesizer) as well as the continuously expanding attack vector of recent GPS receivers. Manufacturers of GPS receivers are adding more configuration and connectivity "features" that normally have security as an afterthought and are easy to break using a custom GPS satellite signal.

1.4 Proposed solution

Our solution is based on modifying a PTP Boundary Clock [Figure 3a]. The modified Boundary Clock [Figure 3b] acts like a normal Boundary Clock but has an additional port that will act as a backup in the case the Grandmaster Clock fails. This backup port will be connected with other backup ports of the various Boundary Clocks distributed throughout the network. To ensure time synchronization among all nodes, we need to have several such GPS based Grandmaster Clocks and various Boundary Clocks around the network to ensure geographical distribution. Instead of using a GPS Grandmaster Clock for each Boundary Clock, the same Grandmaster Clock is able to synchronize various Boundary Clocks as long as the network delays and jitter allow it, reducing the total number of GPS Grandmaster Clocks and as a result, the cost.

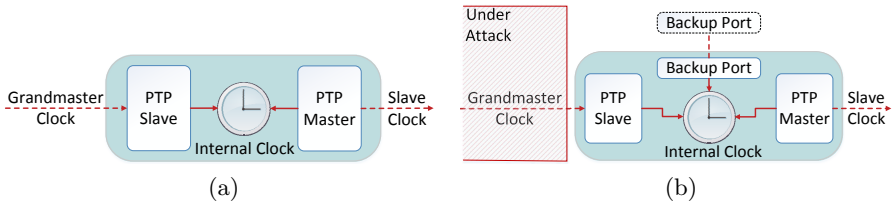


Figure 3: Boundary Clock as originally defined by PTP (a) and with the proposed modification (b).

Not only does our solution manage to reduce the number of GPS receivers needed but it also allows for the detection and mitigation of GPS attacks. This allows the node to always synchronize itself to a time source even if it is not as good as the original.

2 Attack Model

Our focus is on detecting and mitigating attacks on the GPS receiver (i.e., Grandmaster Clock) that offers time information to the boundary clock as well as on the communication link between them. There is an initial prerequisite that **all boundary clocks are synchronized to a correct time source before any attacks occur**.

Attacks against the Grandmaster Clock can cause it to fail in several different ways. We abstract these failures in two classes, as perceived by the Boundary Clock:

- **Crash faults** are observed when the Grandmaster Clock is incapable of sending any type of information to the Boundary Clock. This can happen when the GPS receiver of the Grandmaster Clock is:
  - under a Blocking or Jamming attack and loses its satellite signal lock;
  - under a Software Attack that crashes the GPS receiver;
- **Arbitrary faults** (or Byzantine [8]) are assumed when the information received by the Boundary Clock from the Grandmaster Clock is incorrect in any way, or when it reaches the Boundary Clock late or in an arbitrary order. This can happen when the GPS receiver of the Grandmaster Clock is:
  - under a Spoofing Attack that causes the Grandmaster Clock to give usable but incorrect data;
  - under a Software Attack that renders the Grandmaster Clock unable to provide any kind of usable information.

  A Byzantine fault is also assumed when the Communication Link interconnecting the Grandmaster Clock to the Boundary Clock is under attack, which in turn causes the time information to be delayed or to reach the Boundary Clock in an incorrect order.

3 System Architecture

We define three new port states based on the existing three states of PTP Master, PTP Passive and a PTP Slave. The new port states are called Fault-Tolerant (FT) PTP Master, FT PTP Passive and FT PTP Slave respectively [see Figure 4]. As previously mentioned, our modified Boundary Clock distinguishes itself from a normal one by the presence of one extra port that will interconnect various such Boundary Clocks.

- **FT PTP Slave** has the purpose to detect abnormal activity coming from the Master port it is connected to (in our case this will be the Grandmaster

Clock port) and to alert the backup port. It does this by analyzing the time information coming from the Grandmaster Clock, as well as the network delay between them;
– **FT PTP Master** and **FT PTP Passive** are port states in which the newly added backup port of the Boundary Clock will run. Their primary purpose is to distribute time information (if the Grandmaster Clock where it synchronizes itself is still considered correct) to other Boundary Clocks.

The normal PTP Slave port of the Boundary Clock is replaced with a FT PTP Slave. As mentioned before, the backup port of the Boundary Clock will either run in FT PTP Master or FT PTP Passive modes. A modified version of the Best Master Clock algorithm runs between the backup ports with the responsibility of deciding which one will be the FT PTP Master. After the best FT PTP Master is selected, all the other ports will run in FT PTP Passive. The modified version of the BMC algorithm takes into account another parameter used for disqualifying one such port from the algorithm.

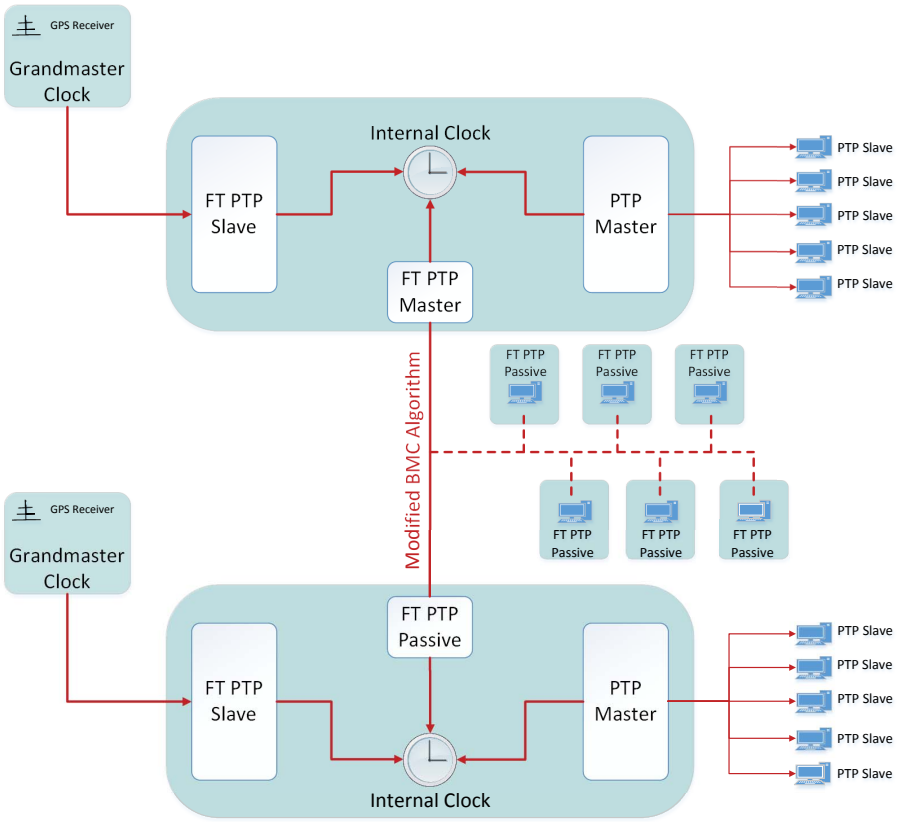


Figure 4: System Architecture

4 Detecting and Recovering from Failures

In this section we describe how the detection and recovery mechanism works. The state of the Boundary Clock during normal operation and when under attack are presented in [Figure 5a] and [Figure 5b].

When an attack is detected by the FT PTP Slave, it first does a clock jump[1] of *2 * DEFAULT_SYNC_INTERVAL* (1 second in our implementation) ahead, then proceed to disable itself. The backup port (running in either FT PTP Master or FT PTP Passive) observes the clock jump by reading the clock at each full run of the protocol. Next, it goes into a normal PTP Slave mode. Any out of the ordinary modifications to the internal clock of the Boundary Clock are detected and handled in this way. The backup port can be in two different states when an attack is discovered: FT PTP Master or FT PTP Passive. If the current state is FT PTP Master, another backup port will need to be elected to go into FT PTP Master state. A modified BMC algorithm is used to perform this change, by comparing data from all the backup ports of the Boundary Clocks.

Notice that we use the local clock of the Boundary Clock as a means to communicate failure detection between the two ports. This is done in order to minimize changes to the standard. However, a low level, high speed communication scheme as presented in [9] can help with the recovery time. The clock jump performed by the FT PTP Slave will always have to be greater than the *DEFAULT_SYNC_INTERVAL* because we need to make sure that the FT PT Master (now in PTP Slave state) will receive at least one *Sync Message* from the new time source (in this case a FT PTP Master port from another boundary clock) to compare the origin timestamp from this *Sync Message* to the current local clock time.

The state change from FT PTP Master to PTP Slave is carried out by modifying the *clock_class* value of this port from 13 (associated with a Master only port) to 255 (Slave only port). Once into PTP Slave mode this port will start synchronizing to another FT PTP Master and the previously set local clock jump will be immediately detected and corrected. In the worst case scenario the recovery time is *DEFAULT_SYNC_INTERVAL + PTP Slave Initialization Time*.

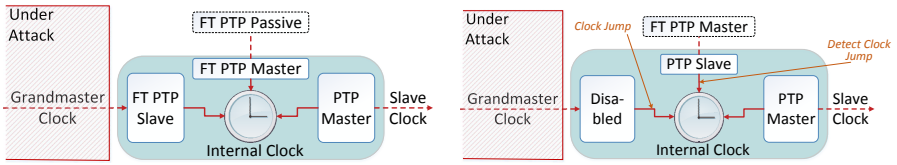


Figure 5: Boundary Clock state during normal operation (a) and after an attack is detected (b)

[1] A sudden change in the order of milliseconds or larger of the internal clock time.

Next we describe in more detail how the detection mechanism implemented in the FT PTP Slave works for crash and byzantine faults.

4.1 Crash failure detection

The crash failure detection mechanism is based on the periodic transmission of *Announce Messages*. The normal behavior of the Grandmaster Clock is to send an *Announce Message* every *DEFAULT_ANNOUNCE_INTERVAL* (1 second in our implementation). The FT PTP Slave keeps track of the received announce messages and waits a maximum of *announceTimeoutGracePeriod* (with a value of 3 seconds) before declaring a packet as lost. We consider the Grandmaster clock failed if two *Announce Messages* are lost. This translates to a maximum wait time of 6 seconds, which is a lot but serves as a means of testing and verifying the algorithm. Of course, the bounds could be made tighter but with the cost of an increased performance overhead.

4.2 Byzantine failure detection

As a first detection layer all information reaching the Boundary Clock from the Grandmaster Clock is filtered to eliminate any bogus data. This will prevent most arbitrary behaviors that cause the transmission of unexpected data. From the previously mentioned attacks, the only one that can do this is the GPS Software attack. This scenario encompasses most of the attack vectors but two cases need to be treated separately because of the their volatile nature:

- **Attacks on time.** In this type of attack the goal is to try to provide inaccurate time information to the Boundary Clock to desynchronize it. The primary information used to detect these attacks comes from *Sync Messages*. An attack can occur in two different ways: either the provided time value jumps forward by a significant amount in one step (clock jump), or the clock is skewed[2] gradually step by step. In order to detect both types, we save the values of the clock offset between the Grandmaster Clock and the Boundary Clock after the clocks are synchronized (Original offset), and the offset calculated in the last execution cycle of the algorithm, which is based on the latest received *Sync Message*. We know that the values will vary very little considering that clocks are already synchronized (prerequisite). So if an attacker tries to jump the clock this will be discovered by comparing the original offset with every new calculated offset. The values used to define the maximum allowed offsets are: *MAX_SECONDS_ALLOWED_OFFSET*, and *MAX_ALLOWED_DRIFT (in ns)*. The first one is used to detect offset jumps in the order of seconds.
  The second one is employed to either find clock jumps by comparing the last known offset with the newly calculated one, or to discover skews by comparing the original offset with the newly calculated one. Special care

[2] A change in the order of μs or slower relative to the internal clock time.

needs to be given to the *MAX_ALLOWED_DRIFT* parameter because it represents the maximum value that the Boundary Clock time is allowed to change at each execution cycle. If an attack increases the clock offset at a rate slower than the *MAX_ALLOWED_DRIFT* parameter, time data from other backup ports can be used to detect it. Two maximum offset values were used because time is represented by two fields (seconds and subseconds field) in our implementation. The *MAX_SECONDS_ALLOWED_OFFSET* and *MAX_ALLOWED_DRIFT* parameters need to be adjusted by measuring and averaging the clock offset and network delays between the Grandmaster Clock and Boundary Clock;

– **Attacks on the communication link.** As previously mentioned the attacker has total control over the communication link between the Grandmaster Clock and the Boundary Clock. This allows him to delay or drop PTP packets. Dropped packets will make the Boundary Clock think the Grandmaster Clock has crashed, and it will act accordingly.
Delay detection is done in a similar way to the detection of Attacks on time but using the *network delay* as primary information. The delay is calculated by means of the standard Delay request-response mechanism [Section 1.2]. Once again we save the first calculated delay value after the clocks are synchronized and the one computed in the last cycle of the algorithm. The main difference consists in the fact that network delays can vary more than the clock offset. Therefore, instead of immediately considering the Grandmaster Clocks failed, we consider deviations from the normal delay to be outliers. For a delay value to be considered an outlier it has to be greater than *MAX_OUTLIER_DEVIATION (in ns)*. When a sufficient number of outliers (*MAX_DELAY_OUTLIERS*) are detected over a period of time, the Boundary Clock will consider the Grandmaster Clock failed and act accordingly.

4.3 Modified Best Master Clock Algorithm

The main modification on the BMC algorithm was the introduction of a new parameter called *isCandidate*. This parameter is verified in each execution cycle of the algorithm and is used to disqualify the FT PTP Master/FT PTP Passive port of the Boundary Clock, when it goes into PTP Slave mode (when an attack is detected). When this change is observed by the remote Boundary Clocks, a new FT PTP Master is elected for the network. In order to exchange the *isCandidate* value between all the Boundary Clocks the *Announce Message* was used.

5 Experimental Evaluation

5.1 Testing Platform

The open source ptpd2 [10] project was used as a base for our implementation on three Ubuntu 14.10 machines. Ptpd2 is specifically tailored to be used as a software only implementation [11] without any hardware assistance.

One of the machines was equipped with two network cards (one for the FT PTP Slave port and one for the FT PTP Master/FT PTP Passive port) and played the role of the Boundary Clock. The other two machines simulated a Grandmaster Clock (running a normal PTP Master) and a second Boundary Clock with a backup FT PTP Master/FT PTP Passive port, to act as a new time source when the first one fails. We did not simulate the other normal PTP Master port of the Boundary Clock and the network it synchronizes because it is just a normal PTP network.

Since we used off-the-shelf PC machines, the internal oscillators of the clocks are of low quality. If left unsynchronized, they drift apart by a large amount in a relatively short span of time. This would not happen in a real smart grid network because the PMU's oscillators are of a better quality. The test network was a normal 100 Mbps Ethernet LAN with a lot of nodes, traffic and jitter. It is safe to assume that the network conditions in a smart grid environment will be considerably better. However all nodes are relatively close geographically, closer than in some deployment scenarios of a smart grid network.

5.2 Results

We experimented our algorithm in three test cases. Our first test case consisted in injecting a Crash fault [Figure 6]. This was simulated by simply killing the PTP process running on the Grandmaster Clock machine. This was done at time "08". After exactly 6 seconds (as previously described *2 * DEFAULT_ANNOUNCE_INTERVAL*) the Grandmaster clock was considered failed. At that moment the FT PTP Slave port of the Boundary Clock increased the internal clock by 2 seconds and proceeded to disable itself. The FT PTP Master/FT PTP Passive port now detects the clock skew and goes into PTP Slave mode. After receiving the first *Sync message* from a FT PTP Master port of another Boundary Clock the synchronization process starts.

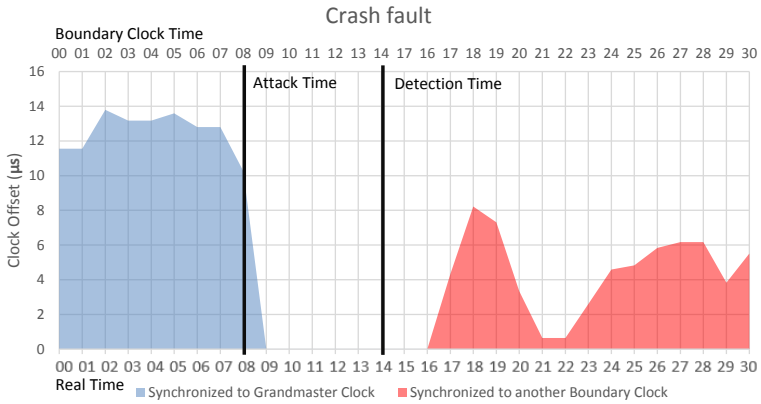


Figure 6: Crash fault detection and recovery

The second test case is an attack on time. This can happen in two ways: either by skewing the clock or by jumping it a large amount at once. We present results for the first case, which is harder to detect. The second case detection scheme is similar except we compare the current local time to *MAX_SECONDS_ALLOWED_OFFSET* instead of *MAX_ALLOWED_DRIFT*. The results would also be similar because nothing else changes. A bash script was created and ran on the Grandmaster Clock machine. It will skew the clock by a small amount by performing a small set of simple operations. The script reads the value of the clock, saves this value into a variable then proceeds to set the clock to the value of that variable. This works because it takes $\simeq$ 1 ms to set a variable in memory in our machines. [Figure 7] shows the perceived increase in the clock offset of the Master (appearing on a logarithmic scale in the graph) that is suffering the attack (left side of the graph). The value for the *MAX_ALLOWED_DRIFT* parameter used for this test was 700 μs.

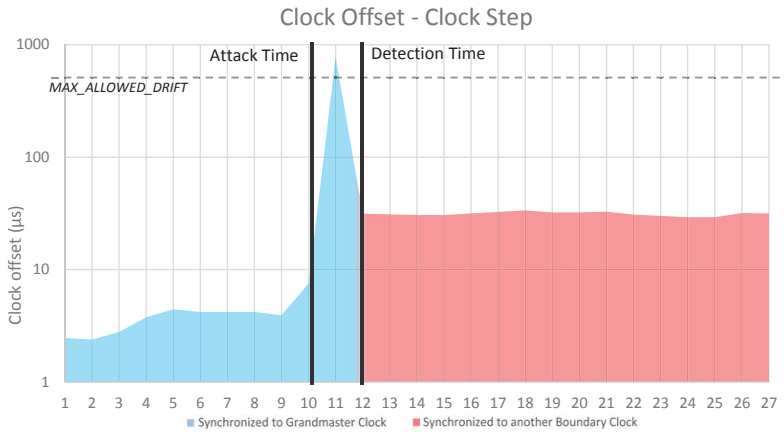


Figure 7: Gradual Clock Skew

The last case we tested was when the attacker delays PTP packets sent by the Grandmaster Clock. The *MAX_OUTLIER_DEVIATION* was set to 300 μs and the *MAX_DELAY_OUTLIERS* was set to 3. To simulating the delays we employed the *tc* command [12]. The *tc* command is used to show or manipulate traffic control settings. This allowed us to introduce a 450 μs delay in the network. We can see from the graph [Figure 8] that the delays of the new communication link between the FT PTP Master/FT PTP Passive (now in PTP Slave state) port of the Boundary Clock and a FT PTP Master port from another Boundary Clock are not as stable as the ones between the Grandmaster Clock and the FT PTP Slave port (now disabled). This is due to the fact that the delays are calculated based on the clock offset and the first Boundary Clock is not yet fully synchronized to the new time source. After some time they will normalize and become stable.

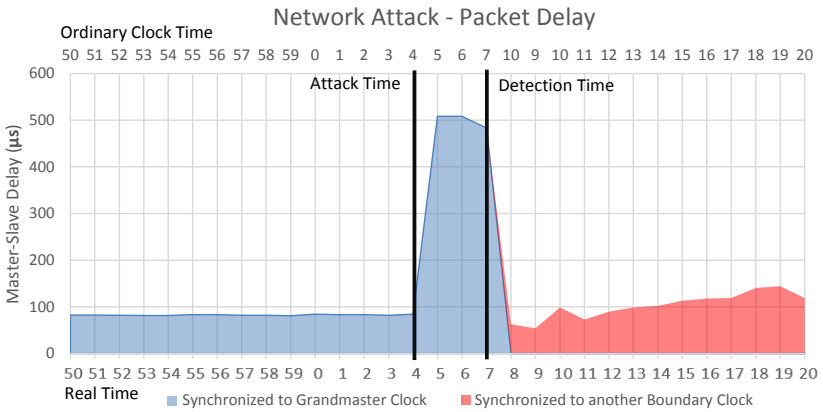


Figure 8: Packet Delay Attack

6 Conclusions

The evolving needs of the smart grid bring a new set of problems that has to be addressed. Time synchronization in such an environment is a critical necessity. This requirement is currently realized by the used of various GPS receivers, which creates a large attack surface for attackers to exploit. In this paper we presented a new approach to using PTP in a smart grid environment while trying to keep the modifications to the standard to a minimum. We have shown that this solution is successful in mitigating all types of attacks as long as our requirements are met. Further research can be done to see how the protocol behaves in a real-world smart grid environment as well as the impact it has on specific parts of the network while under attack.

7 Acknowledgments

This work was partially supported by the EC through project FP7 SEGRID (607109), by national funds of Fundação para a Ciência e a Tecnologia (FCT) through project UID/CEC/00408/2013 (LaSIGE).

References

1. H. Hu and N. Wei, "A study of GPS jamming and anti-jamming," in *Proceedings of the International Conference on Power Electronics and Intelligent Transportation System*, vol. 1, pp. 388–391, Dec 2009.
2. "IEEE Standard for Synchrophasor Measurements for Power Systems," *IEEE Std. C37.118.2011*, Dec 2011.
3. "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pp. c1–269, July 2008.
4. J. Tsang and K. Beznosov, "A Security Analysis of the Precise Time Protocol," in *Information and Communications Security*, Lecture Notes in Computer Science, 2006.
5. T. Mizrahi, "Time synchronization security using IPsec and MACsec," in *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication*, pp. 38–43, Sept 2011.
6. J. Larcom and H. Liu, "Modeling and characterization of GPS spoofing," in *Proceedings of the IEEE International Conference on Technologies for Homeland Security*, pp. 729–734, Nov 2013.
7. T. Nighswander, B. Ledvina, J. Diamond, R. Brumley, and D. Brumley, "GPS Software Attacks," in *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 450–461, 2012.
8. L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, pp. 382–401, July 1982.
9. X.-H. Cheng and L. Zhang, "A research of inter-process communication based on shared memory and address-mapping," in *Proceedings of the International Conference on Computer Science and Network Technology*, vol. 1, pp. 111–114, Dec 2011.
10. *Precision Time Protocol Daemon - ptpd2*, Feb. 2011. http://ptpd2.sourceforge.net.
11. M. B. Kendall Correll, Nick Barendt, "Design Considerations for Software Only Implementations of the IEEE 1588 Precision Time Protocol," in *Proceedings of the International Conference on IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, 2006.
12. *TC Linux User's Manual*, Dec 2001. http://lartc.org/manpages/tc.txt.

On Client-Side Bottleneck Identification in HTTP Servers

Ricardo Filipe, Serhiy Boychenko, and Filipe Araujo

CISUC, Dept. of Informatics Engineering, University of Coimbra, Portugal
{rafilipe, serhiy}@dei.uc.pt, filipius@uc.pt

Abstract. Standing on virtualization techniques, low maintenance costs and economies of scale, cloud computing emerged in the last few years as a major trend in the industry. Since cloud resources grow and shrink as needed, providers and users of the cloud must carefully determine the exact amount of such resources they need. For this reason, getting accurate and timely information from the system is of paramount importance to properly adjust the means serving a given application. However, previous attempts to detect bottlenecks have resulted in complex, heavy and customized frameworks that lack any sort of standardization and may change widely from provider to provider. Improved monitoring mechanisms should be independent from the server technology, should require little to no configuration and should provide information of the real quality of service offered to clients.
To reach these goals, we intend to observe the server infrastructure from the outside and gather the smallest possible number of metrics from the inside. We undertook several experiments in a controlled server, to identify the patterns that correspond to bottlenecks. These experiments clearly show that one can actually diagnose different bottlenecks, by analyzing response times on browsers. These results pave the way to future monitoring mechanisms, mostly based on quality of service evidence, supported by user data.

1 Introduction

In the last few years, cloud computing assumed a role of growing importance in computer systems. Licensing costs, energy, staff wages, the pay-as-you need billing contract, elasticity and the focus on business core rather than infrastructure are some aspects that make cloud computing so appealing. In fact, cloud computing services deliver different types of facilities, with minimal management effort or interaction with the service provider. This interaction occurs via networking and provides access to a shared pool of resources of virtual machines, load balancer, data management facilities, development platforms or even finished software products, depending on the cloud model, which is usually classified as one of three: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS).

Despite being already quite successful, this new paradigm also raises concerns related to trustworthiness and dependability [1, 2]. Since resources are pooled

and shared among many clients, providers might be tempted to under-provide for the services they host, thus causing a poor experience for users, with low performance, excessive delays or even service outages. To prevent service disruption, the cloud provider or the owner of the service must promptly identify and remove bottlenecks, by launching extra resources, such as more bandwidth, CPUs or disk space. Unfortunately, this is not so simple in practice, because the providing side lacks a perfect picture of the service as seen by the client, from mouse clicking to downloading the last byte. Additionally, having precise metrics of a running system is expensive and causes a lot of interference with the system itself.

To properly acquire or release cloud resources, service owners need a clear picture of system performance. This applies to the IaaS model, but may help controlling the quality of a PaaS deployment as well. In both cases, we assume that the client has limited access to the infrastructure or is unwilling to deploy an extensive monitoring solution, but still needs extremely accurate and timely information to control the power of elasticity.

In Section 2, we focus on this exact problem: detecting bottlenecks using the minimum and simplest possible metrics. We aim to perform this bottleneck detection in three-tier web sites using client-side data, because clients have a better perspective of the offered quality of service than providers. To evaluate this possibility, we perform batch submission of requests to the service and collect timing responses on the client. This is not unlike the current paradigm of HTTP performance tools, like HTTPerf [3] or JMeter [4]. With one of these tools, the system administrator controls the invocation of large numbers of requests to observe the response of the system, usually for the sake of fine-tuning performance.

However, since these tools impose a heavy load to the service (and thus are usually ran offline), we aim to perform a similar evaluation online, while standard users are running the service. Instead of generating artificial requests, our goal is to use real requests for the same purpose, by collecting and uploading browser data to the system administrators. Our long-term goal is to identify as many problems as possible. In this paper, we restrict our effort to the three resources that are more important for performance: CPU, server I/O access and client-server bandwidth.

This paper describes our initial efforts to take these client-side measurements: our technological approach, the experiments and the differences we get from measuring everything from JMeter. Interestingly, while in theory we could use JMeter in a controlled environment to distinguish the source of the bottleneck (bandwidth, I/O, or CPU) from one metric alone (total time to obtain the answer), in a real uncontrolled environment we need to split this metric into other two (time to get the first byte, and time to get the entire response) and add one more metric: the query time of the request (or the CPU usage time, which will provide us the same result).

The main contribution of this paper is precisely to demonstrate the feasibility of identifying specific bottlenecks (CPU, I/O or network) using browser

metrics plus an internal server metric. We describe this process and our main contribution in Section 3.

We review the current literature in Section 4. We discuss the current status of our work and future perspectives in Section 5.

2 The Client-Side Tool

We follow a very simple approach to detect possible bottleneck causes of 3-tier Web systems. We only consider three possible causes: processing, database or bandwidth bottlenecks. Processing bottlenecks are related to CPU limitations, which may be due to HTTP thread pool limitations of the Web Server (specially the front-end machines), or CPU machine exhaustion, e.g., due to bad code design that causes unnecessary processing. Database bottlenecks are related to Input/Output operations, which clearly depend on query complexity, database configuration and database access patterns. Bandwidth bottlenecks are related to network congestion, significantly affecting client-server communication times.

We assume the point of the view of an IaaS cloud, although our method can also partly apply to a PaaS or a mixed scheme. The client does not own the resources, but has some control on the source code of a site he or she wants to make publicly available. However, the client may have limited access to the cloud infrastructure configuration. I.e, the client can deploy, but may not be able or willing to change cloud configuration. Hence, we want to avoid capturing a large set of internal system metrics, but we can collect timestamps during the HTTP processing, doing only slight modifications to the application source code.

Table 1. Measured metrics

Metric	Description
Request Time	Time between connection initialization and first response byte received from server
Response Time	Time between first and last response byte received from server
Latency	Time delay experienced in client-server communication
Query processing time	Time that an HTTP request spends on the database
Total Time	Request + Response Time

We initially considered a number of metrics that should allow us to understand if the system has some bottleneck and where. These metrics contain a mix of network and database times. We do not need the CPU time, because we can infer this time based on the query processing time. In Table 1 we list these metrics and their corresponding description. In an ideal closed setting, we only

Table 2. Metrics Required to Detect Bottleneck (Theoretical Results)

	Bandwidth	Database	Threads
Request Time	F	T	T
Response Time	T	F	F

needed the Total Time (request time plus the response time) to distinguish the aforementioned three bottlenecks, if they occur separately. In a bandwidth bottleneck between client and server, the difference between request and response time will tend to grow with the load of the system (specially if the HTTP answer is considerably bigger than the request). In a database bottleneck, the time to get the first byte of response will tend to grow with the load of the database. A processing bottleneck will have a similar effect. To distinguish these two cases, we can submit multiple equal jobs at once using more jobs than threads available to run them: responses will come in groups. To give an example, assume that the server has a pool of 5 threads. If we submit 20 jobs at once that take around 0.5 seconds of CPU time each, every 0.5 seconds (plus a few other delays) we will get 5 different responses. This is a clear indication of a CPU bottleneck. In reality getting such a clearly defined pattern is not so easy, because requests will not occur in clearly defined batches. Hence, we need one additional measurement to distinguish between a CPU and a database bottleneck. We use the query processing time. Unlike the previous metrics of Table 1, this one is internal to the server. We could also consider the processing time, but this is reciprocal to the former. Fortunately, we may eliminate the latency, because this is constant and should not grow on a server bottleneck. In Table 2, we display the relation between the bottlenecks that we are observing and the variables necessary to detect them. We use a "T" (true) and "F" (false) to express the necessity or lack of it to use a variable to observe a given bottleneck. These relations are ideal. A practical setting may be more complex, as we show in Section 3.

A simple way to take measurements from the client side is to use a performance evaluation tool, like Apache's JMeter. The inconvenience of this approach is that it may only work if the client machine is powerful enough to stress the server and the server is disconnected from real users in a testbed.

We intend to follow a different approach. Since it is our goal to tackle more generic settings, we want to use data from real clients. For this we collect the required metrics directly from the user's browser, using the JavaScript Navigation Timing API[1]. This JavaScript library can read the request and the response times of a given HTTP interaction. When the requested resource loading finishes, we send performance indicators (via AJAX) to a small web service, which stores timing data on a database. The remaining times of the process might be kept directly on the server side[2]. Putting in simple terms, a final implementa-

[1] `https://dvcs.w3.org/hg/webperf/raw-file/tip/specs/NavigationTiming/Overview.html`

[2] Nevertheless, for the sake of simplicity, in the tests we performed, we first sent the server data to the client, which then uploads all the metrics in a single operation.

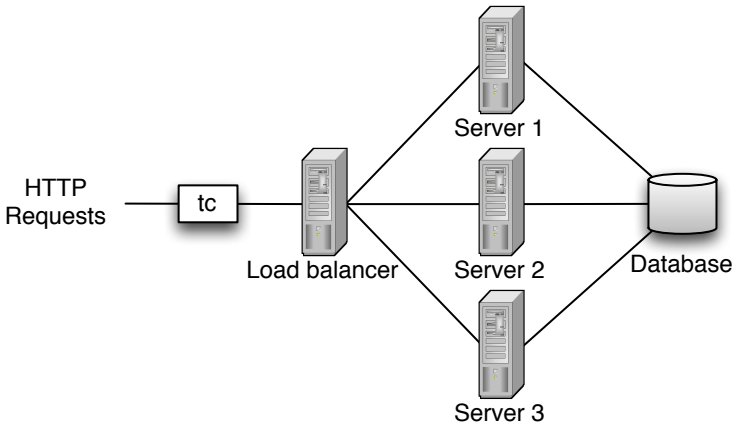


Fig. 1. Experimental setup

tion should works like this: the owner of the site should add a few JavaScript lines to the web page, to instruct the browser to collect the necessary metrics. After collecting these metrics, the browser should send them to the server. This will enable the owners of the site to analyze performance results as seen by the clients. This analysis, however, might be more complex than with JMeter, because it is full of real life noisy data. In the next section, we observe and tackle this precise problem.

3 Experimental evaluation

The goal of our experiments was to observe the feasibility of taking client-side measurements to detect performance bottlenecks. For this, in this Section, we first describe the setup, before detailing the experimental results.

3.1 Experimental Setup

To run our experiments, we deployed the "Java Petstore" [5] application, including the Petstore schematic tables. In the front-end of the server, we have a load balancer that directs user requests to a group of GlassFish Application Servers [6] running in different VMs. These Application Servers take care of the presentation (first) and business (second) tiers. The business tier is stateless, because the Java application keeps all its data in the back-end database (the third tier). This architecture is illustrated on Figure 1. The load balancer machine runs an Apache HTTP Server and an AJP Connector for the load balancing. We also installed a Traffic Control tool (tc)[3] on the system entry point to simulate a congested network. To ensure that some page requests took longer and

[3] `http://www.tldp.org/HOWTO/html_single/Traffic-Control-HOWTO/`.

Table 3. Software used and distribution

Component	Observations	Version
Load Balancer	HTTPD with AJP Connector	2.2
Cluster	VMs with GlassFish	3.1.2
Database	MySQL	5.1.69

to avoid cache utilization that would change the bottlenecks, we increased the contents of the database tables, relatively to the original petstore.

Table 3 summarizes the most important software components of this experiment. We evaluate the performance of this system using two different methods. The first injects requests through a standard performance evaluation tool (Apache JMeter). This approach enables us to run a finely controlled experience, although limitations of the client machine running the Apache JMeter may cause requests to slide in time. To simulate an intensive utilization of the web site by browsers, in the second method, we submitted requests to our infrastructure using the Firefox web browser from 15 machines. The requests were submitted using a script that started the browser on every machine. The purpose of this experience is to analyze the difference between a tool like JMeter and a simulation that is closer to the real-life Internet utilization with different browsers accessing the same infrastructure.

We injected three different bottlenecks on the server: a database bottleneck, a network bottleneck and a CPU bottleneck. The first one corresponds to requests that read a large amount of data from the database, to inject the second one we use the traffic control tool and, to delay the responses in the CPU, we reduce the overall number of threads in the cluster to 5 and put them to sleep 1 second.

3.2 Results

We show the results of our experiments in Figures 2, 3, and 4. On the left-side of the figures we have the response to the JMeter tool. On the right-side, we have the response to the browsers. The x-axis shows the number of the request, from 1 to 50 with JMeter, and from 1 to 15 with the browsers, whereas the y-axis shows the time of each response in milliseconds.

Using JMeter, we managed to minimize the number of external factors interfering with our measurements. For example, network latency is almost always the same, because all the clients are run from the same node, in the same local area network as the server. Furthermore, the tool will spawn multiple threads at once that will send the same request to the server within the shortest possible time frame. This will contribute to a nearly simultaneous arrival of all the requests. In the case of the browsers test, due to the distributed nature of the simulation environment, the results are much noisier. We noticed that the delay of starting a new browser sometimes goes to several hundreds of milliseconds, making it very hard to perform simultaneous requests, not to mention the differences in the local clocks, as each machine decides when to launch the browser. Unlike

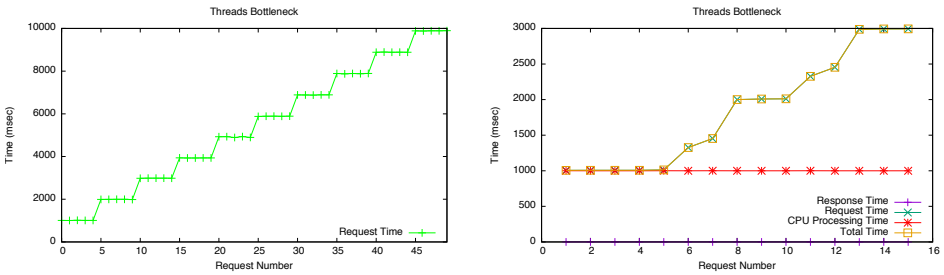


Fig. 2. CPU bottleneck

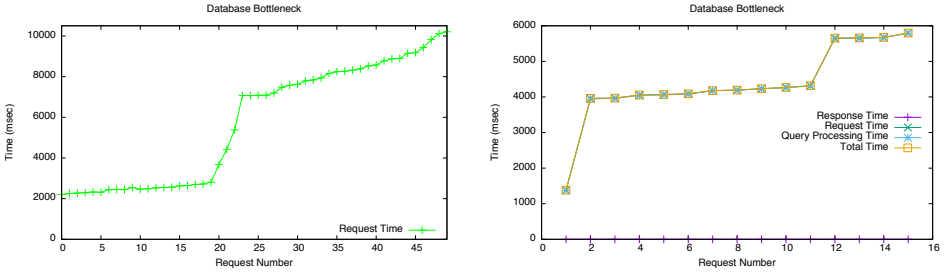


Fig. 3. Database (I/O) bottleneck

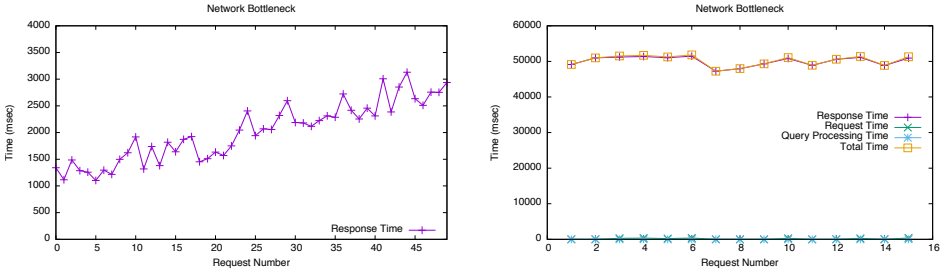


Fig. 4. Network bottleneck

this, in the JMeter case, the delay of the requests was within the few milliseconds range. The browser experiment is therefore more akin to a possible real utilization of the site, but, as we shall see, results become harder to interpret.

After plotting the results, we identified several patterns resulting from Request and Response Times. Firstly, for the JMeter tests, we noticed the ladder-type behavior, when the server reaches its HTTP thread pool limit (in our case, the GlassFish cluster was limited to serve 5 simultaneous requests). Since all clients were launched almost simultaneously and given the significant processing time, the steps of the ladder are easily identifiable on the left side of Figure 2.

We repeated the tests, changing the number of threads in the HTTP thread pool of cluster instances and were always able to observe this behavior.

When a database bottleneck is present in the system, the time taken to generate the page is dominant in the overall process. This behavior turns Request Time into the largest factor in communication, because the time it takes for the server to send the first byte of the response is mostly consumed on the database query. When the bandwidth is the source of the performance problem, the time taken to transfer the page from the server becomes dominant in the total communication time. We can ignore the time it takes to transfer the data from the client to the server, because in most cases the client request size is very small in comparison to the server response. The Response Time is therefore crucial to identify a bandwidth bottleneck in the infrastructure. Additionally, we expect the pattern of bandwidth bottlenecks to be less regular, because the request and response packets have to go through a congested channel. We can see these differences on the left sides of Figures 3 and 4. One should notice that these plots display different metrics. In the case of the network bottleneck, we display the Response Time, whereas in the other two cases we display the Request Time (both metrics are easily available in JMeter and browsers). This observation agrees with Table 2, i.e., these times suffice to identify different types of bottlenecks.

Most of the patterns we observed on the previous experiment occur again in the browsers experiment, but it is harder to distinguish between different bottlenecks using only client-side measurements. For example, the CPU bottleneck loses its ladder-like aspect that is so characteristic in a very controlled environment (refer to the right side of Figure 2). Despite still being there, the effect is much less visible in the browsers experiment, and, we believe that, in general, one might be completely unable to identify this specific kind of bottleneck from the Request Time alone. A clear separation requires an extra variable to distinguish the time the request spends on the CPU from the time it spends on the database. We show the CPU processing time (Figure 2 right) and query processing time for this (Figure 3 right). With these metrics, the component responsible for the delay is immediately identified. In fact, although the difference in patterns between the Request Times of Figures 2 and 3 (which are nearly the same as the Total time) might be unclear, the query processing time is negligible in the case of the CPU bottleneck, whereas the CPU processing time is negligible in the other case. Furthermore, the query processing time we measured includes the waiting time to access the database and thus makes the evaluation simpler, when compared to the CPU processing time, which we can see as a constant in Figure 2 right. In fact, this difference in behavior could be eliminated, if one includes the waiting time for the CPU in the CPU processing time, and, therefore, we can consider the query processing time and the CPU processing time to be pretty much equivalent for our needs.

In light of these results, we are now able to review Table 2, to consider the evaluation of bottlenecks under more realistic settings. As a result, we created

Table 4. Metrics Required to Detect Bottleneck(Practical Results)

	Bandwidth	Database	Threads
Request Time	F	T	T
Response Time	T	F	F
Database Query Time	F	T	T

Table 4, which adds the database query time. This table is a step towards identifying bottlenecks using timing measurements from real web clients.

4 Related Work

We divide the related work regarding bottleneck detection into three main areas: first, we go through academic studies that use data from inside the system to detect bottlenecks; then, we review academic studies that collect information from the client's point-of-view; and, finally, we analyze industrial tools.

4.1 Academic Studies Using Inside System Methodologies

In the literature, we can find a large body of work aiming to detect, predict and even change system configuration, usually in n-tier HTTP server systems, similar to the one we evaluated in this paper [7–12]. For comprehensive purposes, we divided these papers into two main categories: (i) analytic models that collect system metrics, to ensure detection or prediction of bottlenecks; (ii) rule-based methodologies that scale up or down system resources.

Concerning analytic models (*i*), Malkowski *et al.* [13] aim to ensure low service response times. Authors collect many system metrics, like CPU or memory utilization, and correlate them with system performance. This should expose the metrics that best identify the bottlenecks. However, this form of analysis collects more than two hundred system and application metrics. In [14], Malkowski *et al.* studied bottlenecks in n-tier systems even further, to expose the phenomenon of multi-bottlenecks, due to multiple resources reaching saturation. The main conclusion from this work is that lightly loaded resources may be responsible for multi-bottlenecks causing a chain reaction in the n-tier system. The framework used is very similar to their previous work, requiring again full access to the infrastructure. Wang *et al.* followed this approach in [12], with in-depth analysis of metrics in each component of the system. The goal was to detect transient bottlenecks with durations as low as 50 milliseconds. The problem with these approaches is that acquiring such finely-grained data is very hard to transpose to different hardware and software architectures.

[15] presents an approach that uses a queuing model for each tier of the system, to predict the server capacity for a given workload. Authors focused mainly on web server performance and stateless applications. [7] uses DAG-based data flow programs running on cloud infrastructures, to detect CPU and I/O bottlenecks. In [11], authors presented a statistical machine learning framework to

predict bottlenecks and achieve elasticity in terms of VMs. [10] presents a dynamic allocation of VMs based on SLA restrictions. The framework consists of a continuous system introspection that monitors the cloud system and their components. This, however, requires continuous resource consumption (paid by the user) and scalability to large cloud providers. A different approach was followed by [16], where the main goal is not bottleneck detection, but optimal resource utilization using heuristic models.

Regarding rule-based methodologies (*ii*), Iqbal *et al.* [8, 17] propose an algorithm that processes proxy logs and, at a second phase, all CPU metrics of web servers. The purpose is to achieve elasticity concerning the number of instances of the saturated component. [18] uses static performance-based rules. In this approach, if a component resource saturation is observed, then, the user will be migrated to a new virtual machine through IP dynamic configuration.

We can also find many papers that are more closely related to benchmarking and performance analysis than to bottleneck detection. For example, [19], presents the "C-Meter" cloud benchmark framework. It gathers lower level metrics, such as the time spent on resource acquisition, or the wait time in queue for a workload job to be processed. However, these metrics are only accessible to the cloud administrators. [20] benchmarks the Amazon EC2 cloud infrastructure. The main conclusion is the fact that applications in cloud environments will have some performance degradation, when compared to similar application running in Linux clusters and other physical machines. This degradation is strongly associated with the network traffic generated by the application. In other words, applications with higher network requirements will have a higher performance degradation in cloud environments.

4.2 Academic Studies Using Outside System Methodologies

This section reviews solutions that try to detect bottlenecks without having access to the observed infrastructure. In [21], authors propose a client-based collaborative approach. They use a web browser plug-in on each client that monitors all client Internet activity and gathers several network metrics. The plug-in focus is mainly the HTML initial page. It discards page resources from third-party providers, such as CDN objects, and sends all information of the main site to a central point, for processing. The impact of this approach on network bandwidth and client data security is unclear. Additionally, [21] only handles network connectivity bottlenecks.

In [22], authors present Netalyzr, a Java applet for browsers that clients can use to understand why some connection is slow. This tool is mostly used when clients experience some problem. When a client wants to diagnose why some URL is slow, the tool makes several HTTP requests from distinct locations, to gather several network metrics. Although very powerful for connectivity issues, it does not analyze system or performance bottlenecks.

[23] uses a different approach, implementing a Web crawler that gathers HTTP, DNS and TCP connection data from different locations. The main goal is to understand in what network layer do mostly of the user-visible page failures

occur. Dasu [24] is a client-based software. It has more than 90,000 installations, allowing the collection of metrics from different end users. As mentioned by the authors, it is limited by the number of hosts that are online and, consequently, cannot run continuous measurements. It only collects metrics associated with the client network point-of-view, discarding application measurements, such as HTML objects from third-party resources.

In [25], Flach *et al.* present a browser plugin that collects information and analyzes sites based on rules. Again, this paper focuses on network metrics and connectivity issues. Another similar approach is presented in [26], where a Firefox extension based on javascript was created to gather client information, to diagnose network problems. Firelog [27] is another plugin for the Firefox browser. It gathers network metrics for later evaluation of networking issues. Unlike the previous work, the collected data is not processed in real time, but transferred to a PostgreSQL database. Besides connectivity issues, authors also look at client performance problems occurring during page rendering.

In [28], Padmanabhan *et al.* use the PlanetLab [29] infrastructure, to gather network information from 80 sites and analyze the source of the problems. Despite not requiring any browser extension, and thus being able to run out-of-the box (provided that an infrastructure is available), recent studies showed that this pattern of concurrent accesses can significantly change the results observed [30].

4.3 Industrial Tools to Detect and Prevent Bottlenecks

We classify industrial tools for bottleneck prevention into three major groups: (i) monitoring applications that have access to the infrastructure. This group requires heavy configuration and administration, because it is quite similar to the group of applications presented in Section 4.1; (ii) access only to URLs. In this case, the tool allows the client to configure the URLs to monitor and the SMS or email to notify, when something is wrong (according to the tool rules). As in the work we mentioned in Section 4.2, these tools can only detect network bottlenecks or a "slow" server, thus falling short of a deep evaluation; (iii) hybrid tools that collect information from inside and outside of the system. Although very powerful, they need constant maintenance, to ensure the system is correctly monitored [31].

Table 5 illustrates the kind of resource problem detected by some of the aforementioned literature. The second column concerns the need to increase CPU resources or VM instances. The third column is associated to I/O, normally an access to a database. The bandwidth column represents delays inside the cloud network or to the client — normally browser or web services. It is relevant to mention that several papers [7, 14, 32] only consider CPU (or instantiated VM) and I/O bottlenecks, thus not considering internal (between the server components) or external (client-server connection) bandwidth. A different set of work does exactly the opposite, focusing mostly on network issues. In fact, we can clearly see that papers presented in Section 4.1 tend to handle server bottlenecks, whereas papers presented in Section 4.2 tend to cover network bottlenecks.

Table 5. Bottlenecks detection in related work

Article	CPU/Threads/VM	I/O	Connectivity issues
[7]	X	X	
[8]	X		
[13]	X	X	
[9]	X		
[12]	X	X	Internal
[14]	X	X	Internal
[18]	X		X
[21]			External
[22]			External
[23]			External
[24]			External
[25]			External
[27]			External
[28]			External
[26]			External

We are looking for bridging this gap. We are not tied to any specific architecture, as we try to evaluate the bottlenecks from the client side, while, at the same time, we aim to identify precise server bottlenecks. To stay independent from the architecture, we use internal times of the application performing a monitoring task, instead of reading system metrics, like CPU or disk occupation. Furthermore, we try to reduce the collection of the internal times to the smallest number of points, to create a simple system. The fact that we use the standard Navigation Timing API, instead of implementing a full browser plugin also makes things simpler. Additionally, by taking measurements from the perspective of the client, we can have a better insight on the quality of the response. This approach seems preferable to taking a large number of measurements from the distributed architectures involving multiple vendors that compose cloud systems.

5 Conclusion

The allocation of virtualized resources poses a challenge to system architects and administrators as they need to properly provision cloud resources for applications. To achieve this goal, we proposed to detect three types of bottlenecks: processor, bandwidth and I/O. Unlike previous work, we mostly aim to use client-side metrics for detailed observation of the server. The point is to strongly reduce the intrusiveness of monitoring. While other approaches analyze dozens or hundreds of server-side metrics, so far, our evaluation suggests that we need only one such metric, to distinguish CPU time from database query times. Our initial results show that it is possible to interpret the provider infrastructure as a "black box" and still detect bottlenecks.

We recognize that our work is still in an early phase. While this paper demonstrates that we can identify the source of a bottleneck with only a handful of

metrics, the great challenge is to do such detection in real time with actual client requests distributed over time, instead of using a single burst of requests over an offline system. The ability to do so might turn out to be an excellent way of improving the existing monitoring tools, by introducing the client perspective of performance and still distinguishing different types of bottlenecks.

References

1. B.P. Rimal, Eunmi Choi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *INC, IMS and IDC, 2009. NCM 09. Fifth International Joint Conference on*, pages 44–51, 2009.
2. James Hamilton. Internet-scale service infrastructure efficiency. *SIGARCH Comput. Archit. News*, 37(3):232–232, June 2009.
3. Papers — HP Web Server Performance Tool. `http://www.hpl.hp.com/research/linux/httperf/`. Retrieved August 20, 2013.
4. Performance tools — Apache JMeter^{TM}. `http://jmeter.apache.org/`. Retrieved August 20, 2013.
5. Technical white papers — Java Petstore 2.0. `http://www.oracle.com/technetwork/java/index-136650.html`. Retrieved August 20, 2013.
6. Technical white papers — GlassFish Application Server. `http://glassfish.java.net/`. Retrieved August 20, 2013.
7. D. Battre, M. Hovestadt, B. Lohrmann, A. Stanik, and D. Warneke. Detecting bottlenecks in parallel dag-based data flow programs. In *Many-Task Computing on Grids and Supercomputers (MTAGS), 2010 IEEE Workshop on*, pages 1–10, 2010.
8. Waheed Iqbal, Matthew N Dailey, David Carrera, and Paul Janecek. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, 27(6):871–879, 2011.
9. Y. Shoaib and O. Das. Using layered bottlenecks for virtual machine provisioning in the clouds. In *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, pages 109–116, 2012.
10. Nikolaus Huber, Fabian Brosig, and Samuel Kounev. Model-based self-adaptive resource allocation in virtualized environments. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, pages 90–99, New York, NY, USA, 2011. ACM.
11. Peter Bodík, Rean Griffith, Charles Sutton, Armando Fox, Michael Jordan, and David Patterson. Statistical machine learning makes automatic control practical for internet datacenters. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, HotCloud'09, Berkeley, CA, USA, 2009. USENIX Association.
12. Qingyang Wang, Yasuhiko Kanemasa, Jack Li, Deepal Jayasinghe, Toshihiro Shimizu, Masazumi Matsubara, Motoyuki Kawaba, and Calton Pu. Detecting transient bottlenecks in n-tier applications through fine-grained analysis. *ICDCS13*.
13. Simon Malkowski, Markus Hedwig, Jason Parekh, Calton Pu, and Akhil Sahai. Bottleneck detection using statistical intervention analysis. In *Managing Virtualization of Networks and Services*, pages 122–134. Springer, 2007.
14. Simon Malkowski, Markus Hedwig, and Calton Pu. Experimental evaluation of n-tier systems: Observation and analysis of multi-bottlenecks. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 118–127. IEEE, 2009.

15. Rahul Singh, Upendra Sharma, Emmanuel Cecchet, and Prashant Shenoy. Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proceedings of the 7th international conference on Autonomic computing*, ICAC '10, pages 21–30, New York, NY, USA, 2010. ACM.
16. Ruiqing Chi, Zhuzhong Qian, and Sanglu Lu. A heuristic approach for scalability of multi-tiers web application in clouds. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*, pages 28–35, 2011.
17. Waheed Iqbal, Matthew N Dailey, David Carrera, and Paul Janecek. Sla-driven automatic bottleneck detection and resolution for read intensive multi-tier applications hosted on a cloud. In *Advances in Grid and Pervasive Computing*, pages 37–46. Springer, 2010.
18. Huan Liu and Sewook Wee. Web server farm in the cloud: Performance evaluation and dynamic architecture. In *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom '09, pages 369–380, Berlin, Heidelberg, 2009. Springer-Verlag.
19. N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann. C-meter: A framework for performance analysis of computing clouds. In *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, pages 472–477, May 2009.
20. K.R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, Harvey J. Wasserman, and N.J. Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 159–168, Nov 2010.
21. S. Agarwal, N. Liogkas, P. Mohan, and V.N. Padmanabhan. Webprofiler: Cooperative diagnosis of web failures. In *Communication Systems and Networks (COMSNETS), 2010 Second International Conference on*, pages 1–11, Jan 2010.
22. Christian Kreibich, Nicholas Weaver, Boris Nechaev, and Vern Paxson. Netalyzr: Illuminating the edge network. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 246–259, New York, NY, USA, 2010. ACM.
23. C.M. Vaz, L.M. Silva, and A. Dourado. Detecting user-visible failures in web-sites by using end-to-end fine-grained monitoring: An experimental study. In *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*, pages 338–341, Aug 2011.
24. Mario A. Sánchez, John S. Otto, Zachary S. Bischof, David R. Choffnes, Fabián E. Bustamante, Balachander Krishnamurthy, and Walter Willinger. Dasu: Pushing experiments to the internet's edge. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 487–499, Lombard, IL, 2013. USENIX.
25. Tobias Flach, Ethan Katz-Bassett, and Ramesh Govindan. Diagnosing slow web page access at the client side. In *Proceedings of the 2013 Workshop on Student Workhop*, CoNEXT Student Workhop '13, pages 59–62, New York, NY, USA, 2013. ACM.
26. Mohan Dhawan, Justin Samuel, Renata Teixeira, Christian Kreibich, Mark Allman, Nicholas Weaver, and Vern Paxson. Fathom: A browser-based network measurement platform. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC '12, pages 73–86, New York, NY, USA, 2012. ACM.

27. Heng Cui and E. Biersack. Troubleshooting slow webpage downloads. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 405–410, April 2013.
28. Venkata N. Padmanabhan, Sriram Ramabhadran, Sharad Agarwal, and Jitendra Padhye. A study of end-to-end web access failures. In *Proceedings of CoNEXT*, Lisboa, Portugal, December 2006.
29. Papers — Planet Lab. `https://www.planet-lab.org/`. Retrieved June 8, 2015.
30. Joel Sommers and Paul Barford. An active measurement system for shared environments. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 303–314, New York, NY, USA, 2007. ACM.
31. Papers — External Site Monitoring Services. `http://softwareqatest.com/qatweb1.html#MONITORING`. Retrieved June 8, 2015.
32. Balvinder Singh. Article: Bottleneck occurrence in cloud computing. *IJCA Proceedings on National Conference on Advances in Computer Science and Applications (NCACSA 2012)*, NCACSA(5):1–4, May 2012. Published by Foundation of Computer Science, New York, USA.

Analysis of Password Habits and Leaked Databases

Ricardo X. P. Santos[1], Diogo A. B. Fernandes[2], Pedro Tavares[3], Mário M. Freire[1], and Pedro R. M. Inácio[1]

[1] Department of Computer Science, University of Beira Interior, Covilhã Portugal
[2] Portugal Telecom, Portugal
[3] Instituto de Telecomunicações, Portugal
ricardosantos@penhas.di.ubi.pt, diogo-a-fernandes@telecom.pt, ptavares@it.ubi.pt, {mario,inacio}@di.ubi.pt

Abstract Passwords have widespread and critical use in authentication and access control mechanisms of computer systems. However, leakages of passwords or their representations have been making headlines in recent years. This paper describes the analysis of several databases of passwords representations leaked onto the Internet in 2015 and also a study on password habits based on a user survey. The analysis was performed using PassCrackGUI, a toolkit developed in the scope of this work, for several well-known cracking tools, including Hashcat and John the Ripper. Results show that simple dictionary attacks are effective for cracking many passwords and that developers need to employ better storage mechanisms. The results from the survey seem to corroborate previous studies in the conclusion that people often describe stronger passwords when they are asked for.

Keywords: Cracking, Hash, Hashcat, John the Ripper, PassCrackGUI, Passwords, RainbowCrack, Toolkit

1 Introduction

Passwords are critical pieces of authentication in this information era. Many systems and services rely on combinations of usernames and passwords for authentication and access control. Their popularity and widespread adoption is mainly due to their simplicity (in terms of implementation) and ubiquitous nature (any real or virtual keyboard can be used to input the password, easily transmitted over networks). The Internet has largely contributed to this widespread adoption. Unfortunately, passwords may comprise a single point of failure, for systems and services, when compromised. Fernando Corbató himself (the inventor of the password) assumes that passwords are becoming a real problem [23]. The problem is twofold: (i) users choose weak passwords, given the computer power nowadays; and (ii) the passwords need to be safely stored and handled in the systems that use them, and protected when transmitted over networks.

Normally, passwords are not stored directly in computer systems. Instead, one way functions (typically hash functions) are used to created a safe representation from which a password can not be directly recovered [11]. Authentication and access control is done via recalculation of the representation and subsequent comparison with the one that was stored. It is common to combine passwords with salts (random data) for added entropy. Several hashing and Password-Based Key Derivation Functions (PBKDFs) are considered strong enough to produce appropriate password representations. However, these algorithms are not always adopted or correctly integrated and input passwords are usually weak. The number of services and accounts that a user possesses may motivate the choice of weaker passwords, in the sense that they are short, easily remembered, and repeated for different accounts. The leakage and cracking of large password databases [14] often means compromised accounts in more services than the one in which the leak occurred. On the other hand, the cracking tools may be useful in the cases were passwords are lost and documents and accounts need to be recovered.

Password cracking tools are based on calculating the representation of a large number of passwords and comparing them against the hash that was stored [12]. It is even possible to find a string that collides to the same representation different from the original password, if the transformation function is weak. With the increasing interest on password cracking, specialized tools like Brutus [20] (back in 2000), John the Ripper [16], Hashcat [7] or RainbowCrack [18] have emerged. They are expert tools, that can take advantage of multiprocessing on Central Processing Units (CPUs) or Graphics Processing Units (GPUs) for higher performance. Hash functions like the Message-Digest Algorithm 5 (MD5) and the Secure Hash 1 (SHA-1) are not considered safe any longer [5,4], even when using salts.

This paper presents a study conducted over a series of leaked databases containing password representations. The study was performed using a toolkit that integrates several cracking tools, developed in the scope of this work and described herein also. Finally, the results of a survey on password habits, conducted for a population of 64 persons, are discussed towards the end of the paper.

The structure of this paper is as follows. Section 2 describes some password cracking tools and discusses related works. Section 3 briefly describes the developed graphical user interface termed PassCrackGUI. Section 4 includes an analysis on leaked password databases and discusses the results obtained from a survey on password habits. Section 5 contains the main conclusions of this paper.

2 Related Work and Available Cracking Tools

This section respectively discusses related works and available offline cracking tools.

2.1 Related Works

The topics of password cracking and user habits have been scrutinized over the years. Matt Bishop and Daniel Klein, in 1989, started [1] an investigation on password cracking for users of Unix systems. They used `/etc/shadow/` files from people that voluntarily made them available and conducted the study using only the information about the usernames and some available dictionaries. The final results were obtained after a three-year investigation, following several attempts to discover the passwords. In a publication dated from 1995, they claimed to successfully cracked approximately 40% of the passwords, from a total of 14000 different accounts.

A study concerning password habits, which did not included password cracking, was performed in 2013 [13]. It consisted in the analysis of a set of 25000 passwords from the users of the Carnegie Mellon University. They conducted several tests to plaintext passwords to measure their strength. After all tests, authors concluded about some stereotypes, namely that men typically have stronger passwords than women. Other publications have corroborated these findings. They also suggested that informatics and engineering are the professional areas were passwords are better chosen. On the opposite side, areas related with business and policy have weaker passwords. Some of the findings of our study also corroborate their results.

It is common to choose passwords associated with some meaning for the users. They can choose simple words such as pet names, years, or birthdays [8]. Although many studies indicate the use of weak passwords, as the one described herein, the strength of password choices has actually increased over the years [10]. The strength of a password is usually measured according to the entropy (the amount of information present in a bit string). Normally, the higher the entropy, the more seamlessly random, and consequently harder to crack, a password is. Nowadays, online services usually display an entropy gauge next to the password box when registering a new account.

In order to combat and minimize the impact of password breaches, researchers got inspiration from honeypot systems and have proposed decoy mechanisms to be deployed at password storage systems. In the case of [9], *honeywords* were described as a means to add fake and decoy passwords similar to the real one chosen by the user. The generated passwords should be stored in the same location as the real one, but separate from the database where the association between usernames and passwords is made.

2.2 Tools for Password Cracking

There are several offline password cracking tools. PassCrackGUI, the toolkit developed in the scope of this study, integrates three well-known cracking tools [22], namely John the Ripper, Hashcat and RainbowCrack. These tools are briefly described below.

John the Ripper [16] was developed by Alexander Peslyak and provides a free and paid version, which has more functionalities. This tool is usually used for brute-force and dictionary attacks. It can be further highlighted the fact that it

does not require knowing beforehand which hash function was used to obtain a password representation. As foreseen by the Moore's Law, nowadays CPUs pack several computing cores in smaller units. This actually eases brute-forcing and password cracking. With this in mind, Hashcat [7] was mainly built to take advantage of multicore processing units, though John the Ripper also makes use of multithreading. RainbowCrack [18], as the name suggests, makes use of rainbow tables (efficiently precomputed data) as shortcuts for password cracking. Although it still is a method of brute-forcing, rainbow tables speed up the cracking procedure, even for password representations with salts. RainbowCrack was developed by Philippe Oechslin [15] and also takes advantage of multicore processing.

3 Graphical User Interface Functionalities and Implementation

PassCrackGUI is a toolkit integrating several well-known cracking tools (described in subsection 2.2) in a friendly Graphical User Interface (GUI), whose general layout can be seen in Figure 1. The toolkit was developed in Java to favor its portability and was tested in Linux-like and Windows operating systems. The tool is available for the community at GitHub [17] and can be downloaded from there.

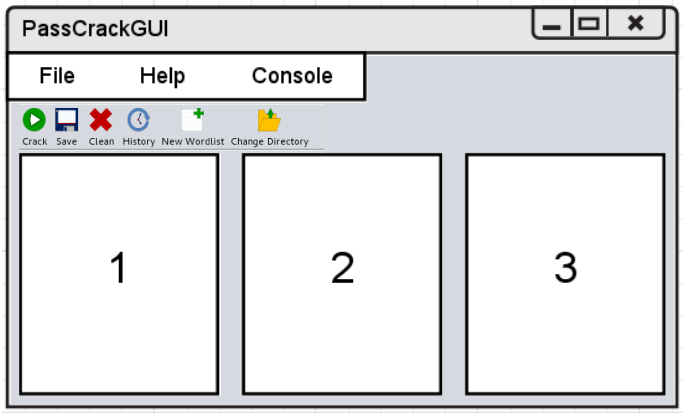


Figure 1. Layout of the GUI of PassCrackGUI.

Apart from the menus, PassCrackGUI is divided into three main areas. Figures 2 and 3 contain screenshots of those areas. In the first area (left side of Figure 2), the user may choose details such as the cracking tool to be employed or the file containing password representations to be cracked. Once initial configurations are performed, the second area (right side of Figure 2) is dynamically

updated to show options specific to each one of the previously selected methods. The third area (see Figure 3) contains two consoles: one for displaying the com-

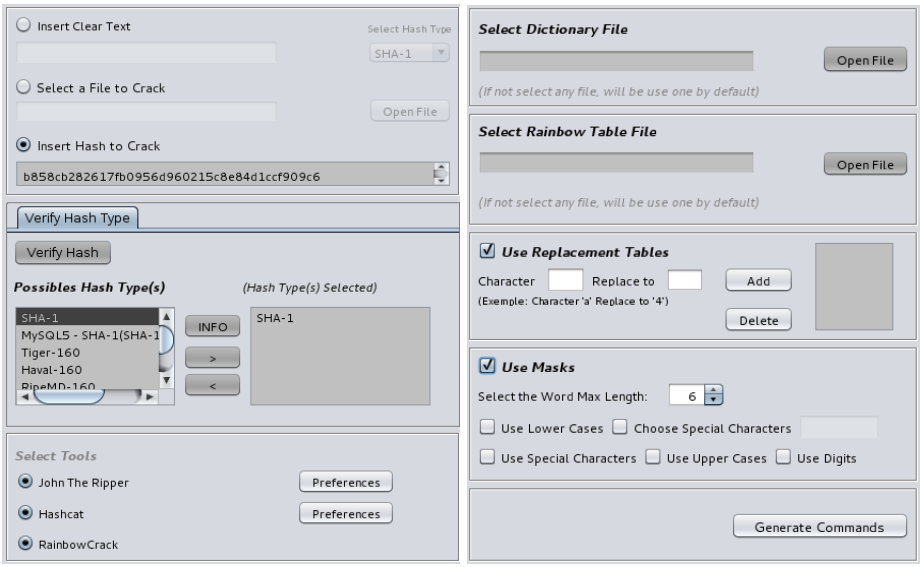


Figure 2. A screenshot of the parts 1 and 2 of PassCrackGUI, where the main configurations of a cracking session are performed.

mands to be executed in the background and another to display the output. The first console can also be used to fine tune the cracking command even further.

The toolkit offers three input cracking modes to the user: (i) a single hash, (ii) a file with multiple hashes, and (iii) a password in plaintext. For the former two modes, PassCrackGUI attempts to crack the specified hashes with the cracking configurations specified by the user. For the latter mode, the tool attempts to crack a hash computed beforehand of the plaintext. If the user knows nothing about the password representations to be cracked, the PassCrackGUI allows to discover that information with the help of Hash ID, a Python script available at [19].

PassCrackGUI makes use of the functionalities provided by the integrated cracking tools, allowing to launch dictionary, brute-force or time-memory trade-off attacks. In addition, it allows to generate a customized dictionary that can be later used in cracking sessions. During execution, the interface saves information and allows to select an attack mode previously chosen.

4 Analysis of Leaked Databases and User Survey

The leaked databases analyzed in the scope of the study herein reported are characterized in Table 1. The table shows the name given to each dataset, the

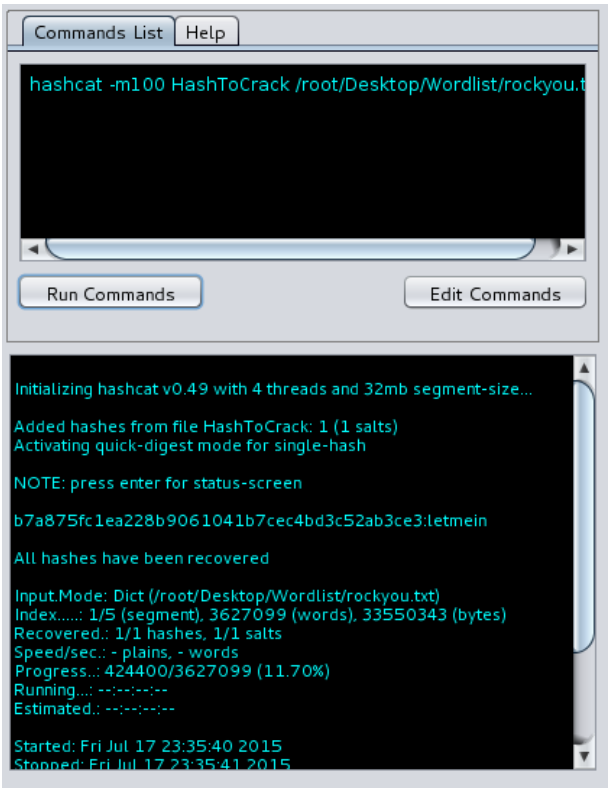


Figure 3. A screenshot of the part of PassCrackGUI with a console interface.

number of password representations contained in each one, and the algorithm used to compute those representations. The exact sources of the leaks are not mentioned for confidentiality reasons, but it can be said that they are from Portuguese systems and were retrieved from popular paste sites, such as PasteBin[4], which are commonly used by hacktivists to exchange or post information online. Some of these leaks concern public institutions and were made public during operations from hacktivists groups in Portugal.

The analyzed databases of password representations were cracked via dictionary and brute-force attacks. For the former case, it was utilized well-know dictionaries such as RockYou [3], HashKiller [2], and others from the Portuguese language. Brute-force attacks were configured to use mask functions, which can achieve more efficient results.

With regard to the user survey, a total of 64 answers were obtained by means of an online questionnaire set up in Google Forms [6]. It was intended with this survey to reach out as many people as possible from distinct areas of knowledge.

[4] `http://pastebin.com`

That was evidently achieved, as illustrated in Figure 4, which shows the number of respondents plotted against the underlying area of knowledge.

	D1	D2	D3
# pass. rep.	63	21	196
algorithm	MD5	MD5	MD5

Table 1. Information regarding databases containing the password representations used in the scope of this work.

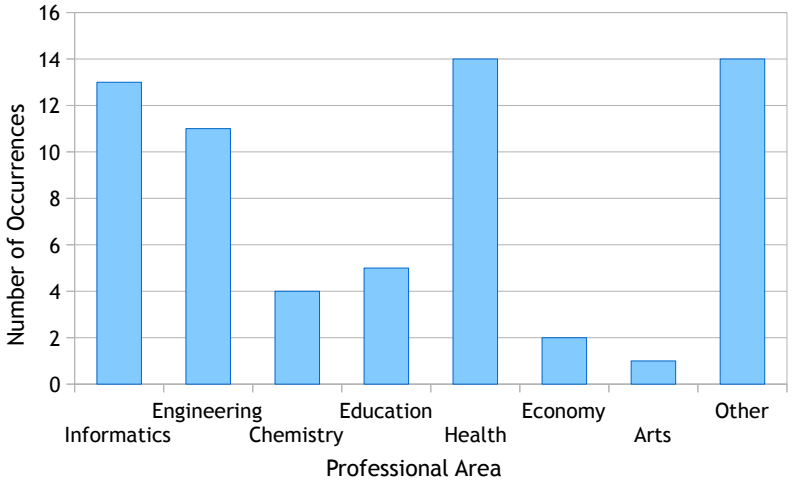


Figure 4. Professional area of the survey respondents.

4.1 Leaked Password Databases

This subsection presents and discusses the results obtained from the leaked databases. A few graphics are included to complement the discussions with passwords statistics deemed relevant with regard to their constitutions.

From the total 280 leaked password representations, 162 were successfully cracked, amounting to 58%, as illustrated in Figure 5. The majority of the passwords were discovered through the use of a dictionary attack. It was found out that most people used simple words from the Portuguese language or Personal Identification Numbers (PINs) of four digits.

Figure 6 contains the percentage of cracked passwords broken down into their lengths. As can be seen in the figure, the length of the passwords varies from three to ten, being four the most popular length. This is due to the extensive

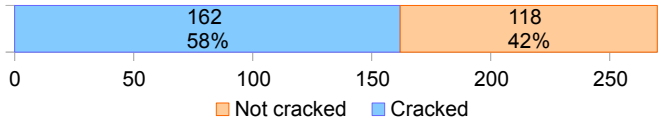


Figure 5. Passwords cracked and not cracked of the leaked databases.

usage of PINs as choices for passwords. Figure 7 includes a graphic demonstration of the proportion of type of characters against their positions in the password. The analysis of the figure reveals that most passwords are composed by lower case characters or digits, or the combination of those groups, and that passwords of length ten usually end with a digit. Few made use of uppercase characters or special characters. This remark is illustrated in Figure 8.

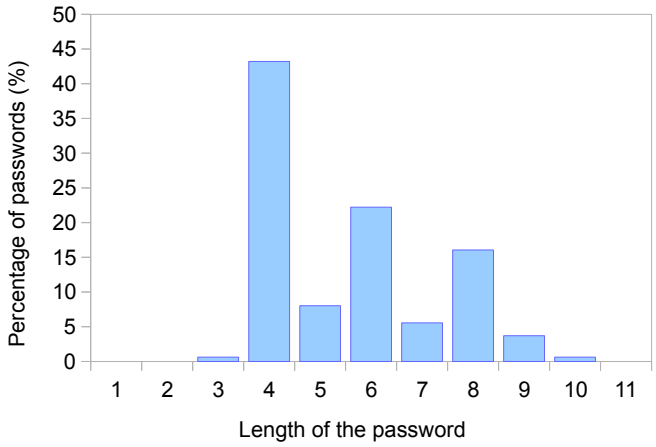


Figure 6. Length of passwords Cracked of the leaked databases.

4.2 User Survey Results

The user survey covered some questions about how people usually construct their passwords, such as inclusion of special characters or digits. The final question asked the volunteers to enter a semantically similar string to their passwords. A total of 64 persons of different demography responded to the anonymous questionnaire. This is illustrated in Figure 9, which plots statistics about the surveyed population with regard to gender and age, which ranged from 15 to 48 years old. There is also a distinct professional area of the respondents, including informatics, engineering, health, among others.

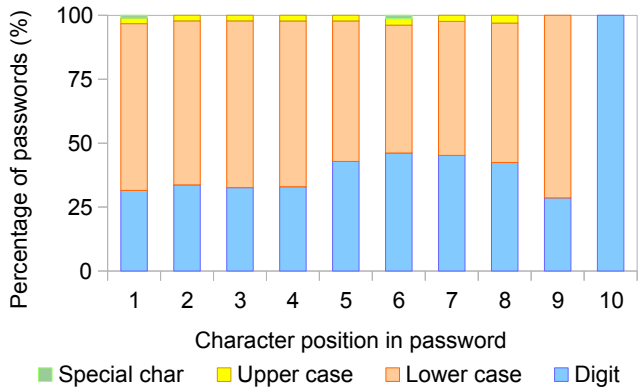


Figure 7. Percentage of different character occurrences with regard to the appearing position in passwords.

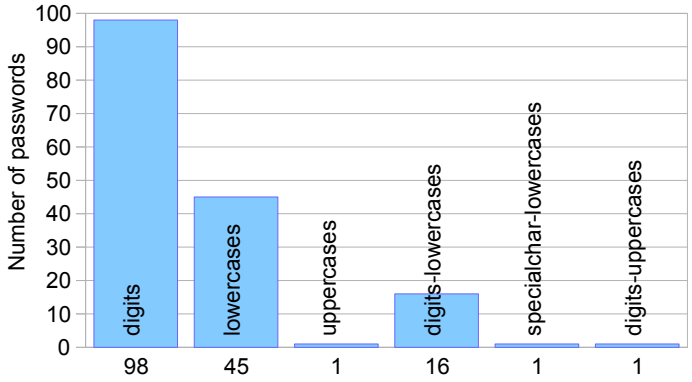


Figure 8. Different compositions of passwords.

As mentioned earlier, the last question of the survey asked the respondents to disclose a string semantically similar to their passwords. The goal of this part of the study is to compute the hashes with the MD5 and attempt to crack the obtained representations. A total of five plaintext passwords from the 64 representations were retrieved. In addition, most claimed to utilize between two and four different passwords with length of eight or more, as illustrated in Figure 10, thereby corroborating the lower number of cracked strings aforementioned. In fact, the maximum lengths of passwords were 65 and 28, perhaps indicating that some users may use passphrases rather than passwords. This is actually a good strategy because passphrases have large amounts of entropy and are easily remembered.

The constitution of the passwords are illustrated in Figure 12. It can be observed that most make use of different types of characters and vary in position.

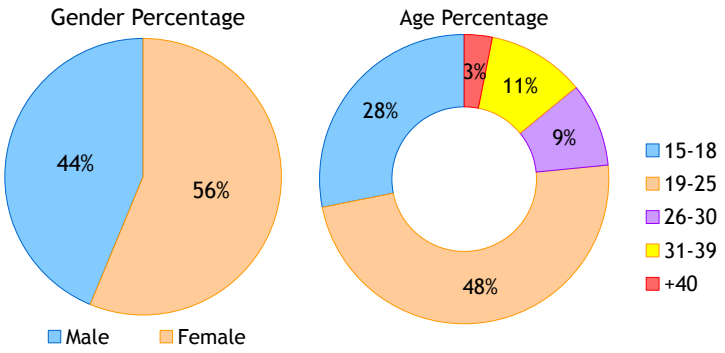


Figure 9. Graphic representing the age and gender percentage of the survey respondents.

However, the second most widely used combination was made of digits and lower case characters.

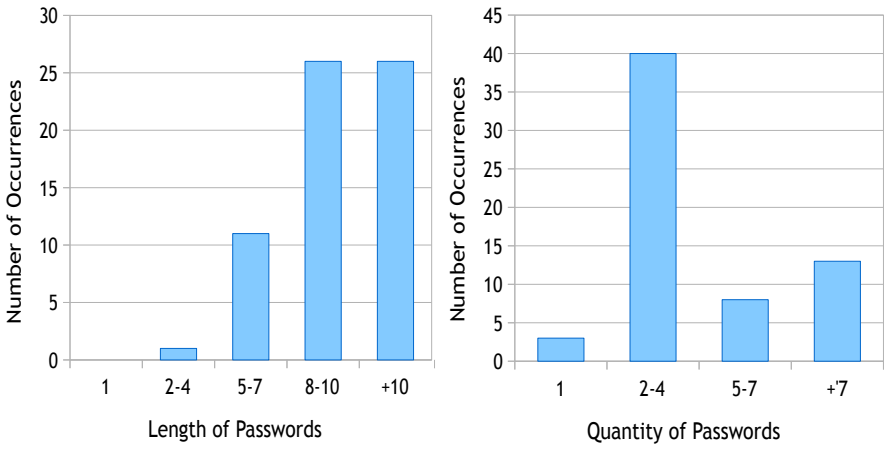


Figure 10. Number of different passwords utilization and length of the survey respondents.

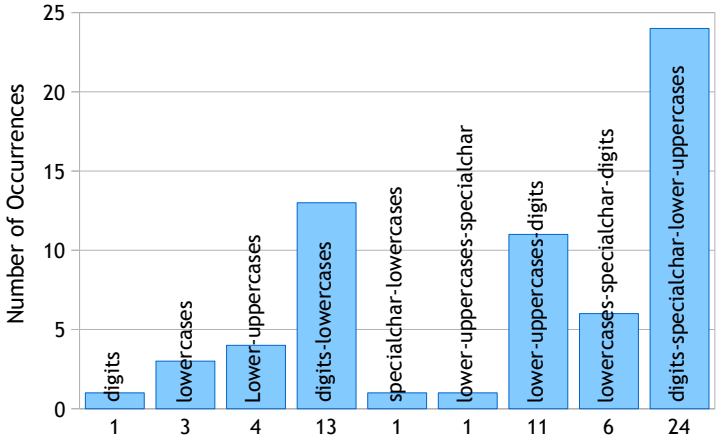


Figure 11. Different compositions of passwords from survey respondents.

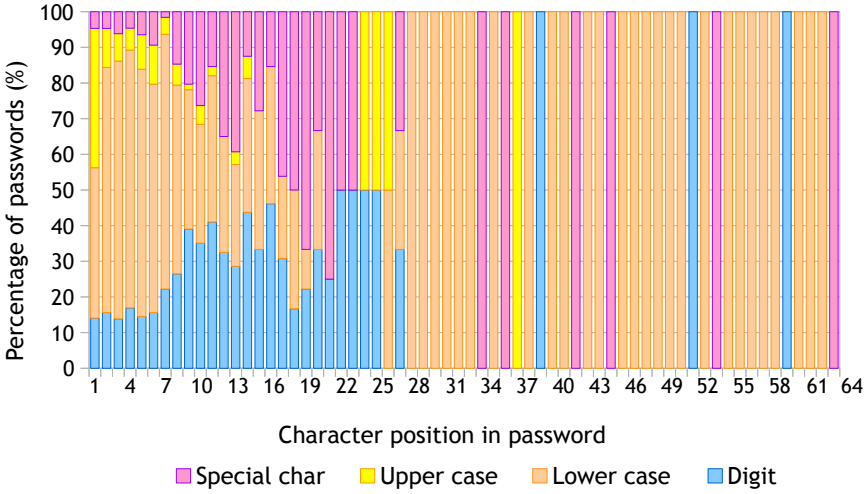


Figure 12. Percentage of the different characters appearances on passwords length position from survey respondents.

5 Conclusions and Future Work

PassCrackGUI allows users not knowledgeable in the area to easily utilize the toolkit and to perform cracking tests (e.g., on his/her own password). The toolkit integrates a set of well-known cracking tools (Hashcat, John the Ripper and RainbowCrack), making use of their functionalities in a concise manner. Several password cracking attacks are currently available, namely dictionary, brute-force and time-memory trade-off (rainbow tables) attacks.

Although most respondents to the user survey were aware of the danger of using weak passwords, it cannot be assumed that they actually use strong passwords to authenticate their accounts. The results obtained from the user survey of password habits are not unanimous. In general, it was concluded from the survey study that users have in general strong passwords. However, it was found out that the leaked password databases are mostly composed of weak password combinations. Our tests achieved a percentage of 58% of successful cracked passwords from all the analyzed databases. This remarking should be understood as an alert for the weak password choices made by users on online services. Moreover, developers should realize that it is required to deploy stronger storage methods, resorting for example to algorithms of the Secure Hash 2 (SHA-2) family or of the more recent Secure Hash 3 (SHA-3) [21].

The results obtained in this study cannot be assumed to be representative. In order to obtain a higher degree of confidence on Portuguese password habits, future work includes studying more passwords databases from the university or companies. Furthermore, it is intended to further develop PassCrackGUI and test it with more configurations using additional computing power from multicore GPUs.

Acknowledgments This work was partially financed by the Fundação para a Ciência e a Tecnologia (FCT), through the UID/EEA/50008/2013 Project.

References

1. Bishop, M., Klein, D.V.: Improving system security via proactive password checking. Computers & Security 14(3), 233 – 249 (1995), http://www.sciencedirect.com/science/article/pii/016740489500003Q
2. Blandyuk: Downloads Page - HashKiller.co.uk (2015), http://www.hashkiller.co.uk/downloads.aspx, last accessed on June 2, 2015.
3. Bowes, R.: Passwords - SkullSecurity (2011), https://wiki.skullsecurity.org/Passwords, last accessed on June 2, 2015.
4. Codahale: How To Safely Store A Password (2010), http://codahale.com/how-to-safely-store-a-password/, last accessed on June 10, 2015.
5. Durmuth, M.: Proceedings of the 2013 workshop on New security paradigms workshop. In: ACM (ed.) Useful Password Hashing: How to Waste Computing Cycles with Style (2013)
6. Google: Google Forms - create and analyze surveys, for free (2015), https://www.google.com/forms/about/, last accessed on June 5, 2015.
7. Hashcat: Hashcat - advanced password recovery (2015), http://hashcat.net/hashcat/, last accessed on June 10, 2015.
8. Hendricks, D.: Social Engineering: What Your Business Needs to Know to Protect Itself (2015), http://www.computer.org/web/computingnow/security/content?g=53319&type=article&urlTitle=social-engineering-what-your-business-needs-to-know-to-protect-itself, last accessed on June 10, 2015.
9. Juels, A., Rivest, R.L.: Honeywords: Making Password-Cracking Detectable. Available in http://people.csail.mit.edu/rivest/pubs/JR13.pdf (May 2013), version 2. Last accessed on June 12, 2015.

10. Komanduri, S., Shay, R., Kelley, P., Mazurek, M., Bauer, L., Christin, N., Cranor, L.F., Egelman, S.: Of Passwords and People: Measuring the Effect of Password-Composition Policies. In: ACM (ed.) Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (2011)
11. Kuznetsov, A.A.: An algorithm for MD5 single-block collision attack using high-performance computing cluster. Cryptology ePrint Archive, Report 2014/871 (2014), http://eprint.iacr.org/
12. Marechal, S.: Advances in password cracking. Journal in Computer Virology 4(1), 73–81 (2008), http://dx.doi.org/10.1007/s11416-007-0064-y
13. Mazurek, M.L., Komanduri, S., Vidas, T., Bauer, L., Christin, N., Cranor, L.F., Kelley, P.G., Shay, R., Ur, B.: Measuring Password Guessability for an Entire University. In: ACM (ed.) Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 173–186 (2013)
14. McCandless, D., Evans, T., Race, V.B., Quick, M., Hollowood, E., Miles, C., Hampson, D.: IWorld's Biggest Data Breaches; Hacks | Information Is Beautiful (2015), http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/, last accessed on June 10, 2015.
15. Oechslin, P.: Advances in Cryptology - CRYPTO 2003, chap. Making a Faster Cryptanalytic Time-Memory Trade-Off, pp. 617–630. Springer Berlin Heidelberg (2003)
16. Openwall: John the Ripper password cracker (2015), http://www.openwall.com/john/, last accessed on June 10, 2015.
17. PassCrackGUI: PassCrackGUI GitHub Repository (2015), https://github.com/PassCrackGUI/PassCrackGUI, last access on June 10, 2015
18. Project, R.: RainbowCrack - Crack Hashes with Rainbow Tables (2015), http://project-rainbowcrack.com/, last accessed on June 10, 2015.
19. r...@blackploit.com: Hash Identifier - Google Project Hosting (2011), https://code.google.com/p/hash-identifier/downloads/list, last access on June 10, 2015
20. Shankdhar, P.: 10 Most Popular Password Cracking Tools - InfoSec Institute (2014), http://resources.infosecinstitute.com/10-popular-password-cracking-tools/, last accessed on June 10, 2015.
21. Stallings, W.: Inside SHA-3. Potentials, IEEE 32(6), 26–31 (Nov 2013)
22. Weir, M., Aggarwa, S., de B. Medeiros, Glodek, B.: Password Cracking Using Probabilistic Context-Free Grammars. In: 30th IEEE Symposium on Security and Privacy. pp. 391–405 (May 2009)
23. Yadron, D.: Man Behind the First Computer Password: ItâĂŹs Become a Nightmare (2014), http://blogs.wsj.com/digits/2014/05/21/the-man-behind-the-first-computer-password-its-become-a-nightmare/, last accessed on June 2, 2015.

Comunicação IP com identificadores criptograficamente gerados para prevenção de DDoS

Ricardo Paula Martins, José Legatheaux Martins, Henrique João Domingos

NOVA LINCS, Departamento de Informática
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
2829–516 Caparica, Portugal
{rmp.martins@campus., jose.legatheaux, hj}@fct.unl.pt
http://di.fct.unl.pt

Resumo Os ataques de negação de serviço (DDoS) representam uma fracção significativa da totalidade dos ataques na Internet e a sua intensidade tem vindo a aumentar. A impossibilidade de na prática autenticar o endereço origem dos pacotes IP é uma das principais razões para o sucesso destes ataques. A solução que consistiria em só utilizar IPSec, TLS e autenticação X.509 não se mostra adequada por ser ineficaz em escala, não ter a adesão completa dos utilizadores finais e não é adequada às ameaças e ataques DDoS.
O sistema de endereçamento IP usado na Internet tem características topológicas e enfrenta diversos problemas de escalabilidade, mobilidade e segurança. Algumas das propostas para lidar com esses problemas passam pelo chamado *"locator / identifier split"*: uma técnica de encaminhamento de pacotes através de túneis, e que usa endereçamento com base em identificadores. Esta técnica lida bem com os problemas da mobilidade e escalabilidade, mas não resolve obrigatoriamente os problemas de segurança do encaminhamento, nem da ausência de integridade do endereçamento IP.
Este artigo apresenta uma proposta baseada em identificadores IP efémeros, gerados com base em técnicas criptográficas, compatíveis com IPv6 e com as as propostas *"locator / identifier split"* do IETF, que introduz um conjunto de mecanismos que permitem garantir a integridade do endereçamento dos pacotes. Assim, torna-se possível detectar a injecção de pacotes maliciosos, com endereços falsos, o que constitui uma primeira barreira para mitigar os ataques de negação de serviço.

1 Introdução

Num artigo muito citado de 1988 [3], David Clark, conhecido pioneiro dos protocolos TCP/IP, reconhecia que a filosofia dos protocolos da Internet sofria de problemas no que dizia respeito à segurança e ao consumo de recursos. Infelizmente, hoje, esse reconhecimento não podia ser mais actual. De acordo com várias fontes [24,12,26,1], os ataques de negação de serviço (DDoS) representam uma fracção significativa da totalidade dos ataques na Internet, e foram recentemente detectados ataques em que as vítimas foram inundadas por tráfego malicioso cujo débito era da ordem de grandeza de 100 Gbps [1]. Muitos dos ataques com maior êxito baseiam-se na utilização de UDP, protocolos aplicacionais sem conexão prévia como ICMP, SNMP, NTP, DNS, ou SSDP, como se

observa na figura 1, e de técnicas de reflexão cujo êxito está dependente da ausência de integridade dos endereços origem dos pacotes [21] e não são contidos pelos mecanismos convencionais de segurança.

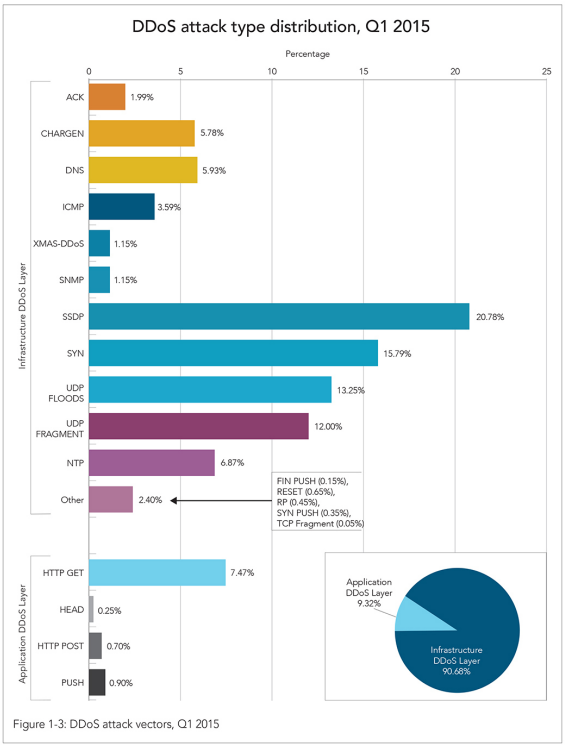


Figura 1: Tipologia dos ataques DDoS recenseados pela Akamai no 1º trimestre de 2015 [24]

1.1 Respostas aos ataques DDoS

Existem muitas respostas simples, directas, mas também erradas, ou pelo menos ineficazes ou irrealistas para este problema. Um grupo de respostas consiste em considerar que se a Internet não fosse tão permissiva e os operadores autenticassem e controlassem eficazmente os pacotes emitidos pelos seus clientes, este problema não existiria. Por um lado não existe motivação económica suficientemente forte para que tal aconteça e o problema permanece. Adicionalmente, uma estrutura de rede baseada em autenticação forte de todos os seus utilizadores pelos operadores, colocaria grandes problemas à entrada de novos operadores, requereria estruturas centralizadas, ou pelo menos muito hierarquizadas de certificação e, no limite, impediria um café, uma universidade, ou um centro comercial, de fornecerem conectividade aos seus utilizadores.

Outro grupo de respostas ineficazes consiste em considerar que a utilização de IPSec, TLS e cadeias de certificados X.509 resolveria todos os problemas. Porém, a prática mostra que estas soluções implicam elevada complexidade de implementação,

processamento e gestão (*e.g.*, IPSec), não escalam, não são adequadas à utilização de endereços efémeros e não defendem de facto o tipo de serviços mais explorados pelos ataques DDoS [27,18]. Estas críticas e limitações de uso têm sido objecto da análise crítica da adopção da normalização IPSec e diversos problemas da sua evolução [27].

No entretanto a maioria das respostas aos DDoS baseia-se em medidas específicas para cada tipo de ataque, numa espécie de competição eterna entre "polícias e ladrões"[1] [21,28]. Importa portanto tentar encontrar mecanismos de segurança gerais para mitigar a eficácia dos ataques DDoS, que mantenham a privacidade a níveis adequados, sem recorrer a soluções complexas, rígidas ou hierarquizadas, ou que introduzam inutilmente latência acrescida.

1.2 Comunicação com base em identificadores

Têm sido avançadas várias propostas de utilização de identificadores de conteúdos, gerados a partir de certificados, que permitem certificar a associação entre o identificador e o conteúdo [7]. Uma ideia semelhante pode ser aplicada aos endereços IP. No entanto, tal solução introduz problemas significativos à localização e ao encaminhamento de e para as entidades em comunicação, visto que os identificadores são gerados por funções criptográficas e perdem as propriedades hierárquicas e topológicas dos endereços IP tradicionais.

Como o *core* da Internet e os sistemas móveis também têm problemas de escalabilidade da localização e do encaminhamento, têm sido propostas soluções, designadas genericamente por soluções *"locator / identifier split"*, ou de mapeamento / encapsulamento (*Map/Encap*) [22], que separam o espaço de endereçamento em dois tipos de endereços (ver secção 2). Os identificadores, independentes da localização das entidades em comunicação, e os localizadores, que são endereços IP tradicionais, associados a infra-estruturas de rede tradicionais. Estas soluções permitem a comunicação entre *endpoints* que usam identificadores gerados criptograficamente [18,17].

1.3 Identificadores efémeros válidos para interacções de transporte

Uma vez que os localizadores resolvem os problemas de encaminhamento, um outro "dogma", que importa descartar, é que os endereços IP desempenham um papel relevante na identificação dos computadores ou dos utilizadores. Mesmo os identificadores propostos pelas soluções *Map/Encap* não passam de identificadores de *endpoints*, e um computador pode usar tantos quantos precise, até vários em simultâneo. Na verdade, esses identificadores só têm um papel relevante para os protocolos de transporte, e só necessitam de ser estáveis enquanto durar uma interacção de transporte de uma aplicação. Vistos deste novo ângulo, os identificadores podem ser efémeros e ligados ao contexto da sua utilização, o que coloca um enfoque muito diferente nos requisitos de segurança, que devem agora ser mais leves e menos exigentes no que diz respeito à autenticidade dos principais, privilegiando sobretudo provas de verificação de integridade contextual das ligações. Apenas os servidores bem conhecidos precisam de identificadores válidos por várias horas, registados no DNS e localizáveis através dos sistemas de mapeamento

[1] Que tem conduzido à proliferação de serviços de DDoS *intelligence*.

de identificadores introduzidos pelos sistemas *Map/Encap*[9]. Uma outra consequência desta abordagem é a possibilidade de atender a requisitos de anonimato, quando interessantes do ponto de vista de salvaguarda de condições de privacidade.

1.4 Proposta de identificação íntegra com baixa latência

A utilização de soluções *Map/Encap* permite-nos conceber uma solução para o problema da integridade do endereçamento dos *endpoints*, baseada na utilização de identificadores efémeros, criptograficamente gerados. A proposta não necessita de introduzir várias rondas prévias entre as partes em comunicação, *i.e.*, é compatível com baixa latência, nem está dependente de terceiras partes, *i.e.*, tem algumas semelhanças com as soluções de autenticação fraca [2]. A solução não pretende substituir as situações em que é necessária autenticação forte, mas apenas introduzir uma nova camada de segurança especialmente pensada para mitigar os ataques de negação de serviço.

A proposta não obriga a nenhuma alteração das aplicações ou dos protocolos de transporte, é de adopção progressiva, e consubstancia-se na introdução de um novo módulo de encaminhamento nos sistemas finais, com propriedades de segurança. A sua adopção permite ao receptor de um pacote inicial de uma "transacção", ou a equipamentos de segurança em seu nome, verificarem que um pacote tem um endereço origem íntegro, e partiu de um emissor que é, com alta probabilidade, uma "pessoa de bem", ou que pelo menos pagou um preço significativo em termos computacionais para o parecer.

As contribuições deste artigo são:

1. A proposta de uma nova visão dos endereços e identificadores que abandona definitivamente a concepção inicial do endereçamento IP, o qual se tem revelado inadequado à mobilidade, escala e introduz insegurança.
2. Uma solução de certificação dos endereços, de baixo custo e baixa latência, completamente *end-to-end* e sem intervenção obrigatória de terceiras partes.
3. A avaliação das suas propriedades de segurança e dos custos computacionais a pagar para delas usufruir ou para as quebrar.

Na próxima secção, secção 2, o artigo apresenta uma panorâmica de algumas soluções *Map/Encap*. Na secção 3 é apresentada a solução proposta e na secção 4 é discutida a sua eficácia no combate a ataques de negação de serviço. A secção 5 avalia os custos de usar esta proposta e de tentar violar as propriedades de segurança dos identificadores. O artigo termina com as secções 6 e 7, sobre o trabalho relacionado, as conclusões e o trabalho futuro.

2 Encaminhamento com base em identificadores

Esta secção apresenta três soluções *Map/Encap*. Duas são normalizadas pelo IETF e agnósticas dos operadores de rede e da estrutura da Internet. Destas, uma ignora o problema da segurança mas a outra não. Uma terceira proposta é suportada num grande envolvimento dos operadores, não é extremo a extremo (*end-to-end*), nem de adopção progressiva, e exige uma nova visão das relações entre operadores.

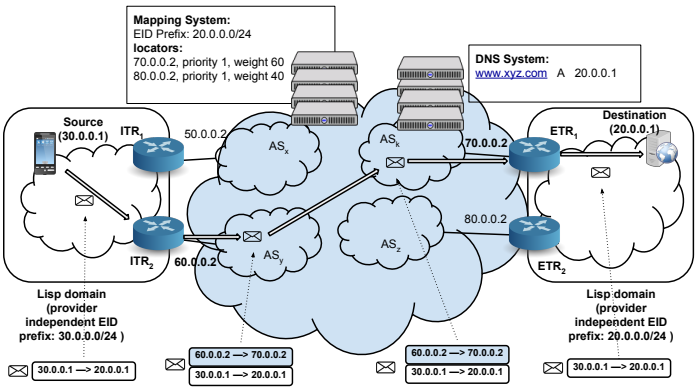


Figura 2: Comunicação entre dois *hosts* usando IDs através de LISP. Os prefixos 30.0.0.0/24 e 20.0.0.0/24 são de EIDs. Os restantes endereços são RLOCs

2.1 LISP - Locator/Identifier Separation Protocol

O *Locator/Identifier Separation Protocol* (LISP) [5,22] é um esquema *Map/Encap*, assente em três princípios:

- separação dos papéis do endereço IP (identificador e localizador),
- mapeamento dinâmico de identificadores em localizadores,
- troca de pacotes entre túneis terminados em *routers* de fronteira.

No LISP existem dois espaços de endereçamento IP distintos: um para os *routers* de fronteira, denominado *Routing Locators* (RLOCs), outro para os *endpoints*, denominado *Endpoint Identifiers* (EIDs). Tanto os RLOCs como os EIDs são compatíveis com os endereços IP actuais, pelo que não exigem alterações ao transporte ou às aplicações. Os EIDs vêm substituir a função de identificador dos endereços IP, enquanto que os RLOCs desempenham o papel de localizador.

Os *hosts* comunicam entre si utilizando os respectivos EIDs em vez dos tradicionais endereços IP, como na figura 2. Os pacotes são trocados sobre túneis estabelecidos entre *routers* de fronteira, sendo encapsulados à entrada e desencapsulados à saída de forma transparente para os *hosts*. O *router* de fronteira de origem encapsula o pacote vindo do *host* num pacote contendo o seu próprio RLOC como endereço de origem e o RLOC do *router* no outro extremo do túnel como endereço de destino. Quando este último recebe o pacote, desencapsula-o e o encaminhamento para o destinatário do pacote original prossegue da forma habitual. Esta solução está integralmente disponível em *routers* Cisco.

O LISP integra um sistema de mapeamento que permite consultar e alterar os RLOCs associados a cada prefixo de EIDs. Este sistema funciona como uma espécie de directoria distribuída, semelhante ao DNS, mas especializada na localização. Apenas os *routers* de fronteira precisam de interagir com os servidores de mapeamento para

actualização dos seus localizadores e obtenção dos localizadores dos outros. Para aumentar a eficiência, os *routers* fazem *caching* das localizações obtidas. O sistema de mapeamento LISP está actualmente operacional e requer autenticação para todas as operações de actualização da localização.

Existe uma variante para a mobilidade, o LISP-MN, em que os *endpoints* desempenham um papel semelhante ao dos *routers* de fronteira, estabelecendo túneis e interagindo com o sistema de mapeamento. Esta solução, muito elegante, suporta a completa mobilidade dos identificadores e dos seus prefixos. O LISP-MN está implementado para Linux e Android através de um módulo de encaminhamento especial.

2.2 SSM - Self-Selectable Mapping

O *Self-Selectable Mapping* (SSM) [16,15] apresenta algumas semelhanças com o LISP, apoiando-se em túneis estabelecidos exclusivamente entre *routers* de fronteira e utilizando um sistema de mapeamento próprio. Contudo, o seu sistema de mapeamento, ao contrário do do LISP, não é acessível publicamente pois apenas os *routers* de fronteira podem fazer consultas e alterações ao mapeamento. Para efeitos de autenticação é usado um sistema assimétrico e as chaves públicas de todos os operadores são conhecidas e replicadas por todos os *routers* de fronteira.

O cliente tem de contratar o serviço de um *Mapping Service Provider* (MSP), que pode ser diferente do seu ISP preferencial ou do para que fez *roaming*. O MSP fornece-lhe a assinatura de um certificado sobre o identificador. Quando o cliente se liga a uma rede, é autenticado e envia o identificador, a assinatura e o nome do seu MSP ao ISP, e os *routers* deste tratam de actualizar a rede onde se localiza. Se se detectar alguma inconsistência, o cliente é rejeitado. Para contactar outro *host*, é necessário conhecer o seu identificador, a assinatura e o nome do MSP do destino e passá-los também aos *routers* do ISP, para que estes localizem de forma segura o destino.

Esta proposta integra as funcionalidades LISP e LISP-MN, complementado-as com um sistema rígido de autenticação e integridade do endereçamento que se afasta significativamente dos princípios *end-to-end* que têm garantido a flexibilidade e a escalabilidade da Internet.

2.3 Host Identity Protocol - HIP

O *Host Identity Protocol* (HIP) [18,17] foi desenvolvido com o objectivo principal de suportar mobilidade. É baseado em alterações ao *host* como no LISP-MN e não exige alterações às aplicações, pois os identificadores são compatíveis com IPv6. Os identificadores, denominados *Host Identifiers* (HI), são a parte pública de uma chave assimétrica auto-gerada. São compatíveis com os endereços IPv6 seguindo a especificação ORCHID [13], em que os primeiros 28 bits são um prefixo ORCHID e os restantes 100 bits são parte do *hash* da chave pública. Esta representação dos identificadores é designada *Host Identity Tag* (HIT).

A chave pública completa, o seu *hash* e servidores de *rendez-vous* (de mapeamento) podem estar publicados no DNS. Além disso, esses identificadores são auto-atribuídos, ao contrário do que acontece com as restantes propostas.

Todas as ligações entre *hosts* que suportem o HIP começam por verificar a identidade do outro extremo, prosseguindo apenas em caso de sucesso. Durante o *handshake* o servidor responde ao cliente com um desafio de dificuldade ajustável, permitindo ao servidor aumentar o custo de início de ligação de forma assimétrica. Todas as ligações são tornadas seguras através de IPsec *end-to-end* e impõem um *handshake* em várias rondas.

2.4 Comparação e caracterização

Entre as soluções *Map/Encap* analisadas, o LISP e o SSM utilizam túneis entre *routers*. O HIP e o LISP-MN são *end-to-end* pois usam túneis entre os *hosts*. O HIP é também o único que utiliza necessariamente identificadores auto-gerados e que pode fornecer garantias de autenticidade (e opcionalmente de confidencialidade).

Quanto ao sistema de mapeamento, o LISP oferece menos segurança que o SSM, que se apoia em autenticação para todas as operações. O HIP também tem algum vantagem sobre o LISP nesse aspecto, uma vez que os identificadores são criptograficamente gerados e auto-verificáveis.

Na tabela 1 apresenta-se uma breve comparação das características mais salientes das soluções analisadas. No caso da proposta SSM, apesar de a publicação[15] não o clarificar, estima-se que as respostas com "?" sejam as correctas. No caso do HIP, a exploração de todas as possibilidades do IPSec permite uma grande flexibilidade de opções de segurança.

Tabela 1: Comparação das características dos esquemas *Map/Encap* analisados

	LISP	**SSM**	**LISP-MN**	**HIP**
Encaminhamento através de túneis entre	*Routers*	*Routers*	*End-to-end*	*End-to-end*
Implica alterações no encaminhamento dos *hosts*	-	-	Sim	Sim
Sistema de mapeamento em servidores e ainda em	*Routers*	-	-	DNS
Integridade dos endereços	-	Sim	-	Sim
Identificador auto-gerado	-	Sim?	-	Sim
Autenticação e confidencialidade do conteúdo	-	-	-	Sim
Confidencialidade da origem	-	-	-	Opcional
Implica alterações nas aplicações	-	Não?	-	-

3 Integridade da origem dos pacotes com base em identificadores efémeros e certificados

A proposta consiste em introduzir segurança na norma LISP, utilizando-a na sua versão *end-to-end* (como o LISP-MN), e na utilização de identificadores LISP efémeros, *i.e.*, válidos no máximo por algumas horas, e concretiza-se em fazer seguir o cabeçalho LISP (com 36 bytes com túneis IPv4), por um cabeçalho opcional de segurança. Logo, pode ser adoptada progressivamente e não exige alterações às aplicações nem ao transporte.

Os identificadores efémeros usados têm o formato de endereços IPv6 e são gerados por técnicas criptográficas como no HIP. O número de bits pseudo-aleatórios pode variar entre 80 (quando identificador tem de ser registado no sistema de *mapping* do LISP) a 121 bits quando tal registo pode ser dispensado e é a seguir denotado por $\#bits$.

A essência da proposta passa por obrigar a parte que toma a iniciativa de iniciar uma "transacção" (*e.g.*, um cliente NTP) com um outro interlocutor (*e.g.*, o servidor NTP), a gerar um identificador origem específico para essa transacção. A parte que recebe um pacote, só o aceita se esse primeiro pacote se fizer acompanhar de um certificado.

Naturalmente, o identificador da outra parte (ID_{dest}), *e.g.*, do servidor NTP, é mais estável, é reutilizado em várias transações com diferentes clientes, deve ser público (*e.g.*, registado no DNS), e está registado no sistema de *mapping* para que os clientes possam encontrar um seu localizador (LOC_{dest})[2]. Este registo confere-lhe um carácter íntegro dado que o sistema de *mapping* do LISP autentica a utilização de um identificador e a actualização da sua localização. No entanto, ID_{dest} também deve ser alterado periodicamente (de 2 em 2 horas por exemplo) para aumentar a robustez do esquema, pois esse identificador vai figurar nos certificados dos clientes.

As entidades envolvidas nas comunicações dispõem de um ou mais pares de chaves públicas / privadas (Pub_{key},$Secret_{key}$) que podem reutilizar à vontade. A certificação das chaves públicas por terceiras partes não é obrigatória e é ortogonal à proposta. Pressupõe-se que as partes em comunicação usam relógios sincronizados globalmente com UTC (*Universal Time Coordinated*) com um erro inferior a $Clock_{error}$.

Para iniciar uma transacção com um servidor conhecido pelo par ID_{dest} e LOC_{dest}, um cliente com o localizador LOC_{orig} gera o certificado $Cert_{orig}$ e depois ID_{orig}:

$$Cert_{orig} = (LOC_{orig}, LOC_{dest}, ID_{dest}, Duration_{ID\_orig}, time, ClientPub_{key}, Sig) \quad (1)$$

$$ID_{orig} = LSB_{\#bits}(HMAC_k(Cert_{orig})) \text{ com } k = \text{SHA-3}(ClientPub_{key}) \quad (2)$$

Sig é uma assinatura gerada a partir de $Secret_{key}$ aplicada aos restantes parâmetros do certificado. Um identificador ID_{orig} é específico para esta transacção, é gerado a partir da informação de contexto incluída no certificado, e só é válido durante $Duration_{ID\_orig} + 2 \times Clock_{error}$ e no momento $time$. O valor máximo de $Duration_{ID\_orig}$ e $Clock_{error}$ são fixados por norma, assumindo-se que valem 10 e 2 minutos respectivamente.

O primeiro pacote do cliente é enviado de ID_{orig} para ID_{dest}, através do túnel LISP (LOC_{orig} / LOC_{dest}). O certificado $Cert_{orig}$ é transmitido no cabeçalho de segurança. O destinatário do pacote realiza as seguintes verificações:

- ID_{dest} é local, frescura de $Cert_{orig}$ e $ID_{orig} = HMAC_k(Cert_{orig})$
- e validade da assinatura Sig de $Cert_{orig}$

Se o pedido for aceite, a resposta segue num pacote de ID_{dest} para ID_{orig} enviado pelo túnel LISP (LOC_{dest} / LOC_{orig}).

[2] O registo no sistema de *mapping* do ID_{orig} é opcional pois o localizador do primeiro pacote do cliente (LOC_{orig}) pode ser usado pelo servidor para a resposta.

Foi feita uma análise de várias opções de suites criptográficas a utilizar que, por falta de espaço, não podemos apresentar[3]. No resto deste artigo serão usadas chaves RSA com 1024 bits, assinaturas RSA-SHA256, HMAC-SHA256-128, com conformidade com o RFC 6379[14]. O correspondente cabeçalho de segurança tem cerca de 350 bytes pelo que não será o responsável directo por fragmentação.

Num ataque por reflexão, usando um protocolo cliente / servidor sobre UDP que envolve apenas a troca de um par de pacotes (que são os ataques mais populares envolvendo endereços origem falsos, *c.f.* secção 1), os membros recrutados para a *botnet* enviam pacotes UDP para os reflectores recenseados usando como endereço origem dos pacotes o endereço IP da vítima.

Com a nossa proposta, o atacante tem de forjar um ID_{orig} igual ao da vítima, usando um certificado que passe o teste $ID_{orig} = HMAC_k(Cert_{orig})$. Essa geração é computacionalmente impossível no tempo de existência do identificador da vítima, mesmo usando um certificado falso, ver a secção 5. Essa propriedade garante que a vítima, ao não reconhecer o ID_{orig} como seu, identifica os pacotes dos atacantes caso estes lhe cheguem.

Adicionalmente, a resposta reflectida (*i.e.*, não solicitada) só segue para a vítima caso passe a verificação da assinatura. Assim, ou o atacante gera uma chave diferente para cada pacote, ou os certificados válidos exibidos têm uma chave pública em comum. Nesse caso a utilização de um mecanismo de *shaping* sobre a chave pública permite limitar o número de reutilizações da mesma chave em novas transacções (*e.g.*, algumas vezes por segundo por exemplo). Este *shaper* é genérico e, tal como todos os outros mecanismos propostos, pode ser implementado por um módulo único, complementar ao encaminhamento LISP. Em serviços críticos, as vítimas preferenciais, todas as verificações referidas podem ser delegadas em equipamentos de segurança que conheçam os *IDs* válidos no seu perímetro de segurança.

A necessidade de verificação da assinatura permite realizar um ataque de negação de serviço usando ID_{orig} que satisfaçam $ID_{orig} = HMAC_k(Certificado)$ mas gerados com certificados forjados, dado que o custo de calcular o HMAC é cerca de 1/3 do de verificar a assinatura. Este assunto será objecto do trabalho futuro referido na secção 7.

4 Análise de desempenho perante um ataque DDoS

Nesta secção considerar-se-à um atacante em posse de uma *botnet* com 10000 *bots* e coordenação centralizada, o que de acordo com um relatório da Symantec [6] constitui uma *botnet* mediana. O ataque será com reflexão, o factor de amplificação[4] é um parâmetro, e explora-se um protocolo baseado em UDP.

[3] As análises experimentais incluiram a avaliação do custo computacional de várias opções criptográficas para se aferir do impacto no processamento: geração e verificação de certificados, incluindo geração de pares de chaves RSA (1024 ou 2048 bits), DSA (1024 bits), ECC (160 e 224 bits); geração e verificação de assinaturas dos certificados (com todas as parametrizações indicadas); bem como custo de geração e verificação dos identificadores com os vários esquemas MAC alternativos.

[4] O termo "amplificação" deve-se à possibilidade do tamanho da resposta poder exceder em várias vezes o tamanho do pedido inicial.

O modelo utilizado foca-se na largura de banda dos pacotes que efectivamente chegam à vítima e com os quais esta terá que lidar. De modo a evitar *bias* na análise a favor desta proposta, consideram-se as seguintes hipóteses pessimistas:

- a largura de banda do ataque está limitada apenas pelos *bots* e não pelos reflectores,
- todos os *bots* participam simultaneamente no ataque,
- o atacante dispõe de tantos reflectores quanto necessite,
- os *bots* estão perfeitamente coordenados e podem partilhar informação entre si,
- os pacotes enviados pelos *bots* utilizam ID_{orig} e certificados válidos, pois os *bots* reutilizam chaves durante o ataque,
- não existem outros mecanismos de detecção activos para além dos descritos.

Este modelo, apesar de minimalista, representa o cenário mais desfavorável de ataque cuja intensidade é, no essencial, proporcional à dimensão da *botnet*.

Optou-se por utilizar este modelo em oposição a uma análise apoiada em *traces* porque não foi possível obter atempadamente *traces* que fossem simultaneamente recentes, representativos de vectores de ataque populares, e publicamente acessíveis.

De acordo com um relatório de 2014 da Akamai [25], a *global average peak download speed* é 26,9 Mbps. Assume-se que a largura de banda de *upload* média de um *bot* é um décimo desse valor, *i.e.*, 2,69 Mbps, e que cada pacote enviado ao reflector tem 100 bytes. Daqui resulta que cada *bot* pode enviar até 3363 pacotes por segundo, e a largura de banda agregada do ataque é 26,9 Gbps.

Para que os valores calculados reflictam a existência de amplificação, multiplique-se a largura de banda calculada por um factor de amplificação A igual à razão entre o tamanho da resposta amplificada e o tamanho do pedido enviado pelos *bots* (100 bytes).

Tabela 2: Características de um ataque DDoS em diversos cenários.

	Sem defesa	Atacante ingénuo	*Round-robin*
Pacotes por segundo totais (pps $\times 10^4$)	3363	10	747
Largura de banda agregada (Mbps)	26900	80	5978

Na tabela 2 apresentam-se três cenários de ataque e suas consequências: ataque sem contra-medidas de defesa, ataque ingénuo e ataque *round-robin*. Estes ataques são caracterizados da seguinte forma:

Sem defesa Neste cenário sem qualquer protecção, o atacante encontra-se limitado apenas pela largura de banda acessível aos *bots*, uma vez que o custo computacional de gerar um pacote é negligenciável e os *bots* não são filtrados pelos reflectores. A largura de banda do ataque deverá ser multiplicada por A (factor de amplificação).

Atacante ingénuo Neste cenário, acrescentam-se três hipóteses: a) o protocolo é seguido por todos os intervenientes, b) os certificados utilizados têm cerca de 350 bytes, e c) todos os reflectores aplicam um *shaper* que admite 10 pacotes por segundos com a mesma chave pública. O atacante diz-se "ingénuo" porque cada *bot* utiliza o mesmo reflector durante todo o ataque.

Os custos computacionais para o atacante não são significativos: cada *bot* gera um pacote válido e reutiliza-o enquanto este for válido e, portanto, apenas tem de gerar um certificado válido (i.e., calcular assinatura e HMAC) uma vez pela duração do mesmo. Considerando uma duração de 14 minutos, e que se demora na ordem de grandeza de 1 ms a calcular uma assinatura RSA com chaves de 1024 bits, e menos do que isso a calcular o HMAC, o custo associado à geração do certificado válido é insignificante.

Assim, o ataque fica limitado pelos *shapers* aplicados pelos reflectores, e a capacidade agregada do ataque é 80 Mbps, a multiplicar por A, o factor de amplificação.

Round-robin Este cenário é uma extensão do anterior, e considera um atacante menos ingénuo que dissemina um subconjunto da lista de reflectores aos *bots*. Estes irão percorrer a lista em *round-robin* de forma a contornar o *shaper* de cada reflector e assim maximizar o número de pacotes reflectidos para a vítima.

Para que a taxa de envio de um bot (V_{envio}) não seja limitada pela taxa do *shaper* (V_{shaper}), basta que a lista de reflectores seja suficientemente longa para ser percorrida na totalidade a uma velocidade menor ou igual à permitida pelo *shaper*. Ou seja, o tempo necessário para percorrer a lista deve permitir espaçar os pacotes enviados para cada reflector tal que esteja abaixo do limite imposto pelo *shaper*.

A taxa de envio de pacotes é reduzida para 747 pacotes por segundo devido à presença do certificado.

Contudo, a resposta do reflector continua a ter o mesmo tamanho que anteriormente, *i.e.*, 100 bytes, verificando-se uma contracção na largura de banda do ataque. Para obter o mesmo volume de ataque que anteriormente, o atacante agora precisa de investir 4,5 vezes mais largura de banda.

Considerando um taxa de envio de 747 pps, uma lista de reflectores com 10^4 elementos (como na hipótese) é percorrida em 13,4 segundos. Se cada *bot* enviar um pacote a cada reflector antes de mudar, cada reflector vê um pacote vindo de cada *bot* uma vez a cada 13,4 segundos, muito abaixo do limite imposto pelo *shaper*. Na verdade, basta que $V_{envio} \leq N_{reflectores} \times V_{shaper}$. Neste caso, o atacante necessita apenas de 75 reflectores para maximizar a capacidade do ataque. A capacidade do ataque neste cenário é de 5978 Mbps, a multiplicar por um A.

Para esta evasão ser eficaz, é necessário que cada *bot* use uma chave diferente dos restantes. O custo da geração dessa chave é amortizado durante a duração do ataque.

Tanto neste cenário como no cenário anterior, a vítima do ataque pode distinguir rapidamente quais os pacotes que fazem parte do ataque, visto que estes terão ID_{dest} diferente dos seus, e é computacionalmente impraticável para o atacante gerar pacotes válidos com ID_{dest} igual ao da vítima, como discutido na secção 5.

5 Custos computacionais de usar o mecanismo e de tentar violar as suas propriedades de segurança

De acordo com o anteriormente descrito na secção 3, para a geração e validação de identificadores utilizam-se entre 80 e 121 bits resultantes das funções HMAC e CMAC e as suites criptográficas do IPSecv3 (RFC4301 [11]) com IKEv2 (RFC5996 [10]), bem

como o RFC 6379[14].

Análise do custo computacional. Os resultados a seguir apresentados são os que implicam o pior caso para o defensor (ou seja o receptor) e o melhor caso para o potencial atacante DDoS (ou seja o emissor), na combinação de suites criptográficas que menos agrava a assimetria entre o atacante e o defensor. Recorde-se que esta combinação é a seguinte:

1. Assinaturas RSA PKCS-1, com chave de 1024 bits e utilização de síntese SHA-256,
2. HMAC-SHA256-128, com o identificador gerado a partir dos 121 bits mais significativos do valor de saída desta função (que calcula inicialmente 128 bits por uma operação XOR de duas partes (direita e esquerda) do output inicial de 256 bits).
3. Nesta configuração o tamanho de um certificado $Cert_{orig}$ é de 350 bytes como explicado antes.

Nas avaliações experimentais utilizou-se a biblioteca OpenSSL, v0.98za, numa plataforma Intel Core i7, 2,5GHz QuadCore, com sistema de operação OS X Mavericks. Nesta plataforma, as operações de exponenciação / multiplicação de inteiros em RSA e DSA já fazem uso de co-processamento híbrido de aceleração em software / hardware.

Custo de geração de um certificado por um potencial atacante O tempo médio observado correspondente a 12.000 gerações de certificados (sem contar com geração de pares chave pública e privada, já que estas podem ser pré-calculadas por parte de um potencial emissor atacante) foi de 0,47ms em média. A sua validação por parte do receptor implica um custo de 0,037ms, dada a assimetria a favor do defensor das operações de assinatura e verificação quando se usa RSA, PKCS#1 e síntese SHA-256. Notar que o atacante só pode gerar o certificado após obter o identificador da vitima, que apenas é válido por algumas horas.

Custo de geração do identificador ID_{orig} com base na função HMAC Nas observações realizadas (que consistiram em gerar e verificar 12.000 identificadores, em 12 séries de 100 identificadores em diferentes instantes nos computadores usados), o atacante consegue gerar em média um identificador em cada 13,98 μs, com o defensor a gastar praticamente o mesmo tempo - média de 13,81 μs para o validar.

Nas séries de avaliações experimentais teve-se o cuidado de proceder à minimização de efeitos colaterais do processamento, minimizando os processos em execução no sistema alvo de medida (isto para além do número de amostras obtidas em séries de avaliações realizadas com intervalos variáveis ao longo de um dia), com obtenção de desvios padrão inferiores a 1% sobre os valores médios medidos.

Análise de segurança. De acordo com os resultados experimentais obtidos, apresenta-se de seguida uma avaliação sobre a efectividade da integridade dos identificadores (com resistência a eventuais colisões que o atacante pretenda explorar, dentro da janela de validade do identificador destino, que como se referiu é fixada em algumas horas).

Para o efeito partimos da avaliação teórica da propriedade de defesa face a colisões fortes (*second-image resistance*) nas sínteses subjacentes à função HMAC. Isto porque vamos considerar (para pior caso do defensor) que este aceitaria o identificador origem

sem um reconhecimento prévio da assinatura do certificado do emissor, o que constitui mais uma vez o melhor caso para o atacante e o pior para o defensor. Note-se que a função HMAC só pode ser calculada pelo atacante depois de obter o identificador destino válido, pois este é usado na implementação como chave partilhada de entrada para o cálculo HMAC do identificador origem que será depois reconhecido pelo receptor a partir dos restantes dados do certificado do emissor.

Uma vez que até ao momento não são conhecidas vulnerabilidades de criptoanálise, comprovadas ou publicadas, em relação à construção da função "sponge" e síntese primitiva Keccak (subjacentes à normalização SHA-3 [20]), assumimos que o oponente só encontrará "no melhor caso" uma possível colisão, quando gerar no mínimo 2^{128} identificadores[5]. Deve ter-se em conta que a colisão tem que ser encontrada sobre um certificado antes considerado válido pelo receptor e no tempo de vida do seu identificador (ou seja algumas horas).

Assim, admitindo que o atacante terá que conseguir realizar no mínimo 2^{128} ($\approx 3,4 \times 10^{38}$) operações de geração de identificadores, tendo por base um certificado que vai apresentar ao receptor e que foi por ele gerado previamente. Isso exigirá ao atacante dispor de $1,32 \times 10^{24}$ horas para o fazer com eventual sucesso. Se por hipótese, o atacante usar uma capacidade computacional equivalente a 10^{10} processadores como o usado no teste, ainda assim necessitaria de $1,32 \times 10^{14}$ horas (ou seja cerca de $1,5 \times 10^{10}$ anos). É claro que ainda podemos considerar que no esquema utilizado apenas se utilizam os 121 bits mais significativos do valor da computação HMAC e não os 256 bits úteis. De qualquer modo, a probabilidade de ocorrer uma colisão dos primeiros 121 bits em 256 bits gerados pseudo-aleatoriamente é aproximadamente igual a 10^{-32}. Este valor pode ser calculado aproximadamente usando a fórmula $1 - e^{-(N*N-1)/2^K}$, com N=256 bits e K=121 bits, cuja dedução completa pode ser encontrada em [23], apêndice 11.

6 Trabalho relacionado

Em [28], Zargar *et al.* categorizam os mecanismos de defesa contra ataques DDoS consoante o ponto onde estes actuam: na origem, no destino, na rede, e híbridos. Já Peng *et al.* [21] consideram quatro categorias com base no tipo de defesa: prevenção, detecção, identificação da origem, e reacção/combate ao ataque.

Os mecanismos baseados na origem podem ser demasiado estritos (*Ingress/Egress Filtering*) e/ou sofrer de falta de incentivos para a sua utilização (quem beneficia não é que implementa o mecanismo). Os mecanismos baseados no destino tendem a ser mais realistas e utilizados porque estão sob o controlo da vítima ou actuam próximo desta, mas tendem a ser específicos para um protocolo ou aplicação, ou dependem de alguma cooperação da rede (*IP Traceback*), ou são fáceis de iludir com endereços de origem falsos. Os mecanismos baseados na rede envolvem alterações à mesma. Os mecanismos híbridos requerem cooperação de vários pontos (detecção no destino e combate na origem, por exemplo), e como tal tendem a não ter sucesso.

[5] Está-se a assumir o uso de HMAC parametrizável com SHA-256 (SHA-3), de acordo com o RFC 6379, a mesma construção usada na mais recente normalização IPSecv3, para a qual não foi encontrada até agora qualquer vulnerabilidade.

É preciso navegar no espaço de mecanismos possíveis e equilibrar os *trade-offs* de especificidade/generalidade e impacto/realismo. O que se propõe é um mecanismo geral e complementar ao que já existe, permitindo uma solução eficaz e eficiente quando combinado com aproximações baseadas em *shaping* e filtragem próximos do destino.

A nossa proposta insere-se nas categorias de prevenção e detecção. Por um lado reduz a largura de banda efectiva dos ataques, por outro permite detectar facilmente se um pacote reflectido é legítimo ou não, bastando verificar se ID_{dest} é o esperado. Para que o atacante consiga o mesmo volume de tráfego de ataque que anteriormente precisa de aproximadamente 5 vezes mais *bots* e reflectores.

Os esquemas *Map/Encap* analisados na secção 2 apresentam filosofias distintas quanto à segurança.

O HIP tem alguns mecanismos de protecção contra DDoS, incluindo desafios e criação de uma ligação autenticada e cifrada sem estado na primeira fase. A especificação para apoio à mobilidade e *multi-homing* [19] descreve também um mecanismo de créditos proporcionais à quantidade de dados enviados por um *host*, de modo a evitar assimetrias excessivas. Os identificadores que utiliza são as chaves públicas dos *hosts*. Em resumo, o HIP é, do ponto de vista do DDoS, uma solução que vai muito para além do necessário, implicando um custo inaceitável em alguns contextos (*e.g.*, estabelecer um canal cifrado e autenticado para enviar um pacote ICMP é excessivo).

O LISP, por seu lado, não oferece qualquer protecção explicitamente contra DDoS.

O SSM não permite contactar *hosts* sem a informação necessária (*i.e.*, identificador, MSP e assinatura). Embora se possa argumentar que isto dificulta a obtenção de *bots* e reflectores, o facto é que esta é feita primariamente com o apoio de *malware* [4,8]. O sistema de *mapping* é resistente a registos falsificados mas poderá ser vulnerável a DDoS devido ao custo das verificações. Além disso, os autores afirmam ser possível utilizá-lo para detectar um ataque DDoS através de um volume anormal de consultas. Em suma, o SSM é, do ponto de vista do DDoS, uma solução que repousa no sistema de *mapping* para a detecção de ataques e é, no máximo, um mecanismo auxiliar que não permite sequer à vítima discriminar os pacotes do atacante.

7 Discussão e trabalho futuro

O endereçamento e o encaminhamento IP enfrenta diversos desafios ao nível de escalabilidade e mobilidade que têm vindo a ser endereçados pelas propostas *Map/Encap*, como por exemplo a norma LISP. Este artigo propõe um conjunto de mecanismos de segurança que estendem a norma LISP de forma a que seja possível verificar, com baixa latência e custo reduzido, a integridade do endereçamento IP. O objectivo é proporcionar uma primeira resposta aos desafios colocados pelos ataques DDoS realizados ao nível rede, que frequentemente se baseiam na utilização de técnicas de reflexão e amplificação, endereços IP falsos e protocolos de transporte sem conexão.

A proposta baseia-se na utilização de identificadores IP efémeros, gerados criptograficamente, mas que podem ser encaminhados na Internet através da norma LISP, e no uso de certificados assinados com chaves assimétricas. Os mecanismos propostos são simples, gerais, facilmente verificáveis, não provocam aumento da latência, são

de adopção opcional e gradual, são independentes dos níveis transporte e aplicação, e permitem a identificação do tráfego de atacantes que usem endereços IP origem falsos.

Essa detecção faz-se com total segurança e êxito junto da vítima, mas também parcialmente nos nós reflectores do ataque. Apesar de os pacotes enviados pelo atacante serem com absoluta segurança facilmente identificados como falsos, e poderem ser filtrados por equipamentos de segurança próximos da vítima, é desejável ir mais longe. De facto, usando *botnets* maciças e bem coordenadas, é possível enganar os reflectores e continuar a bombardear a vítima com pacotes.

O trabalho futuro a desenvolver compreende a extensão dos mecanismos de base apresentados neste artigo com um conjunto de extensões que permitam uma resposta gradual e com custos proporcionais face aos riscos envolvidos. Essas extensões serão constituídas por:

1. um mecanismo de desafio que impeça os atacantes de utilizarem localizadores falsos e bloqueie imediatamente o ataque nos reflectores, impedindo também um ataque por negação de serviço dado o custo de verificação da assinatura, e
2. um sistema de créditos (associados ao par chave / ID_{orig}) com base em capacidades, integrado com autenticação fraca de fluxos de mensagens (*e.g.*, para suporte de TCP ou de sucessivos pedidos baseados em UDP).

No entanto, estas extensões devem continuar a manter as propriedades mais interessantes da proposta actual: simplicidade e implementação facilmente verificável, adopção progressiva materializada na introdução de um novo módulo de encaminhamento IP, possibilidade de integração num TPM (*Tamper Proof Module*), baixa latência, independência dos protocolos de transporte e aplicação, manutenção da privacidade e suporte de autenticação fraca.

Referências

1. D. Anstee, J. Escobar, C. Chui, and G. Sockrider. Worldwide Infrastructure Security Report. Technical Report Volume X, Arbor Networks, 2015. SR/WISR2014/EN/0115-LETTER.
2. J. Arkko and P. Nikander. Weak authentication: How to authenticate unknown principals without trusted parties. In *Security Protocols*, pages 5–19. Springer, 2004.
3. D. Clark. The design philosophy of the darpa internet protocols. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 106–114, New York, NY, USA, 1988. ACM.
4. E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proceedings of the USENIX SRUTI Workshop*, volume 39, page 44, 2005.
5. D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. The Locator/ID Separation Protocol (LISP). RFC 6830 (Experimental), Jan. 2013.
6. M. Fossi, G. Egan, K. Haley, E. Johnson, T. Mack, T. Adams, J. Blackbird, M. K. Low, D. Mazurek, D. McKinney, and P. Wood. Symantec Internet Security Threat Report — Trends for 2010. Technical Report Volume 16, 2011.
7. A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker. Naming in content-oriented architectures. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, pages 1–6. ACM, 2011.

8. J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon. Peer-to-peer botnets: Overview and case study. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 1–1, 2007.
9. M. Hoefling, M. Menth, and M. Hartmann. A survey of mapping systems for locator/identifier split internet routing. *Communications Surveys Tutorials, IEEE*, 15(4):1842–1858, Fourth Quarter 2013.
10. C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996 (Proposed Standard), Sept. 2010. Obsoleted by RFC 7296, updated by RFCs 5998, 6989.
11. S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), Dec. 2005. Updated by RFC 6040.
12. M. Kührer, T. Hupperich, C. Rossow, and T. Holz. Exit from Hell? Reducing the Impact of Amplification DDoS Attacks. In *USENIX Security Symposium*, 2014.
13. J. Laganier and F. Dupont. An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers Version 2 (ORCHIDv2). RFC 7343 (Proposed Standard), Sept. 2014.
14. L. Law and J. Solinas. Suite B Cryptographic Suites for IPsec. RFC 6379 (Informational), Oct. 2011.
15. H. Luo, Y. Lin, H. Zhang, and M. Zukerman. Preventing ddos attacks by identifier/locator separation. *IEEE Network*, 27(6):60–65, 2013.
16. H. Luo, H. Zhang, and M. Zukerman. Decoupling the design of identifier-to-locator mapping services from identifiers. *Computer Networks*, 55(4):959–974, 2011.
17. R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. RFC 4423 (Informational), May 2006.
18. P. Nikander, A. Gurtov, and T. R. Henderson. Host identity protocol (HIP): Connectivity, mobility, multi-homing, security, and privacy over IPv4 and IPv6 networks. *Communications Surveys & Tutorials, IEEE*, 12(2):186–204, 2010.
19. P. Nikander, T. Henderson, C. Vogt, and J. Arkko. End-Host Mobility and Multihoming with the Host Identity Protocol. RFC 5206 (Experimental), Apr. 2008.
20. N. I. of Standards and Technology. *DRAFT FIPS PUB 202: SHA-3 standard: Permutation-based hash and extendable-output functions*. NIST, May 2014.
21. T. Peng, C. Leckie, and K. Ramamohanarao. Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Comput. Surv.*, 39(1), Apr. 2007.
22. D. Saucez, L. Iannone, O. Bonaventure, and D. Farinacci. Designing a Deployable Future Internet: the Locator/Identifier Separation Protocol (LISP) case. *IEEE Internet Computing*, 16(6):14–21, Nov. 2012.
23. W. Stallings and L. Brown. *Computer Security Principles and Practice*. Pearson, 3rd edition, 2014.
24. A. Technologies. State of the internet security report - q1 2015, 2015. [Online; accessed June 2015].
25. J. Thompson, M. McKeay, B. Brenner, J. Sun, R. Möller, M. Sintorn, and G. Huston. Q4 2014 State of the Internet Report. Technical Report Volume 7, Number 4, Akamai, 2014.
26. Verisign. Verisign Distributed Denial of Serice Trends Report - Q2 2014. Technical Report Issue 2, Verisign, 2014.
27. C. H. J. Wu and T. Liu. Simulation for intrusion-resilient, DDoS-resistant authentication system (IDAS). In *Proceedings of the 2008 Spring simulation multiconference*, pages 844–851. Society for Computer Simulation International, 2008.
28. S. T. Zargar, J. Joshi, and D. Tipper. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *Communications Surveys & Tutorials, IEEE*, 15(4):2046–2069, 2013.

Cifra Multimodal Indexável para Aplicações Móveis baseadas na Nuvem

Bernardo Ferreira, João Leitão, and Henrique Domingos

Nova University of Lisbon / NOVA LINCS
Faculdade de Ciências e Tecnologia, 2829-516 Caparica, Portugal
bernardof@acm.org,{jc.leitao,hj}@fct.unl.pt

Resumo Neste artigo propomos CMI, um middleware distribuído de Cifra Multimodal Indexável, que permite que aplicações móveis guardem e pesquisem eficientemente os seus dados multimédia (ou multimodais) na nuvem com garantias de privacidade. Para tal, utilizamos repositórios multimodais partilhados que podem ser acedidos por múltiplos utilizadores em simultâneo. No centro do nosso middleware usamos uma nova família de algoritmos criptográficos, a que chamamos Codificações Conservantes de Distância (CCD), que permitem a transmissão segura de características vectoriais dos dados multimodais para a nuvem. Através destas características codificadas, computações pesadas de indexação passam a poder ser geridas pelos servidores na nuvem, com garantias de privacidade. Mostramos que a nossa solução permite uma melhor gestão dos recursos dos dispositivos móveis, com impacto medível no desempenho das operações e no consumo de energia dos mesmos, quando comparada com as soluções do estado da arte.

Keywords: *Outsourcing* de Dados e Computações, Processamento de Dados Cifrados, Pesquisa de Informação Multimodal

1 Introdução

Actualmente os dispositivos móveis permeiam o quotidiano, correspondendo já a 30% do tráfico global na internet e ultrapassando as vendas de PCs e Laptops em seis vezes [1]. A crescente popularidade dos dispositivos móveis e tablets tem também mudado a forma como os utilizadores produzem e manipulam os seus dados. Por um lado, os utilizadores produzem atualmente maiores quantidades de dados multimodais (i.e. dados contendo diferentes conteúdos multimédia, como fotos, audio e texto), através dos seus dispositivos móveis omnipresentes. Por outro lado, os utilizadores esperam poder aceder e partilhar os seus dados de forma ubíqua.

Devido às suas limitações de recursos (poder computacional, bateria e capacidade de armazenamento) e devido aos cada vez maiores conjuntos de dados multimédia produzidos e acedidos pelos utilizadores, os dispositivos móveis têm sido um importante factor de crescimento de soluções de *outsourcing*, como as baseadas em armazenamento e computação na nuvem. No entanto, guardar e

processar os dados dos utilizadores na nuvem leva a desafios de privacidade. Isto é uma observação natural, consequente de que fazer *outsourcing* de dados e computações também implica fazer *outsourcing* do controlo (e muitas vezes autoria) sobre os mesmos [2]. Por um lado, notícias recentes têm provado que privacidade não é preservada em serviços na nuvem (ex: [3,4] e o recente ataque ao Apple iCloud [5], onde eram guardadas fotos privadas de milhares de utilizadores, incluindo celebridades, geradas por iPhones). Por outro lado, quando se usa serviços na nuvem, é necessário não só ter em conta hackers da internet, explorando possíveis vulnerabilidades de software e hardware na infra-estrutura dos provedores da nuvem [5], como também administradores de sistemas maliciosos ou simplesmente pouco cuidados, que têm acesso a todos os dados em memória principal e secundária nos servidores da nuvem [6–8].

A solução comum para lidar com estas questões de privacidade é usar esquemas de cifra ponta-a-ponta, onde os utilizadores são responsáveis for cifrar todos os seus dados antes de os enviar para a nuvem [9]. No entanto, tais esquemas limitam as operações que os utilizadores conseguem fazer com os seus dados, como operações eficientes de partilha de dados com terceiros confiáveis através da infra-estrutura na nuvem, e operações eficientes de pesquisa sobre esses mesmos dados. Enquanto o primeiro desafio é fácil de resolver com partilha de chaves criptográficas e autorização de acessos, o segundo desafio é não trivial. Apesar de existirem vários trabalhos no estado da arte que tentam endereçam este problema [10–15], tanto quanto sabemos não há nenhuma solução multimodal capaz de suportar simultaneamente qualquer formato multimédia, e as soluções que existem (maioritariamente focadas em texto [11–14], e algumas poucas em imagens [10, 15]) são computacionalmente demasiado pesadas para dispositivos móveis.

Neste artigo propomos uma nova arquitectura middleware para endereçar os problemas anteriormente descritos. Chamamos à nossa proposta CMI, um middleware distribuído de Cifra Multimodal Indexável, uma vez que: (i) permite o armazenamento dinâmico e a pesquisa de dados contendo múltiplos conteúdos multimédia em repositórios na nuvem, com garantias de privacidade; e (ii) permite mover computações de indexação (mostradas mais tarde serem as computações mais pesadas nas alternativas do estado da arte) para a infra-estrutura da nuvem também com garantias de privacidade. No centro do design do CMI encontra-se uma nova família de primitivas criptográficas, a que chamamos Codificações Conservantes de Distância (CCD), que permitem codificar dados de forma segura enquanto preservam uma função de distância controlável entre os dados em claro. Ao apenas requerer que os utilizadores extraiam (e codifiquem com CCD) algumas características vectoriais relevantes dos seus dados multimodais, enviando as mesmas de seguida para indexação na nuvem, CMI é capaz

de superar as limitações de trabalhos anteriores do estado da arte com melhores garantias de desempenho, tornando-se na primeira proposta a ser capaz de comprovadamente suportar clientes móveis de recursos limitados.

2 Trabalho Relacionado

Pesquisa sobre dados cifrados é hoje em dia um tópico de investigação popular, com o recente sucesso de serviços de armazenamento e computação na nuvem e os problemas de segurança que advêm [2, 4, 5]. Nos últimos anos, importantes avanços foram conseguidos nas áreas de criptografia homomórfica [16] e *Oblivious RAM* [17], que permitiriam resolver todos os problemas referidos. No entanto, mesmo as soluções mais recentes [16, 17] continuam a ser apenas de interesse teórico, pois são várias ordens de magnitude mais complexas e pesadas que as usadas em qualquer aplicação prática.

De forma a conseguir um balanço entre eficiência e segurança, a comunidade propôs várias alternativas baseadas em Cifra Simétrica Pesquisável (CSP) [10–15]. Generalizando, estas soluções exigem que os utilizadores construam nos seus dispositivos um índice dos seus dados, sendo este cifrado com uma mistura de criptografia convencional e determinística e guardado na nuvem juntamente com os dados cifrados. Tais soluções conseguem assim tornar aceitável a complexidade adicional imposta pela segurança, com a contrapartida que permitem que sejam reveladas ao provedor da nuvem alguns padrões de acesso aos dados. Nomeadamente, revelam os chamados padrões de pesquisa (se esta pesquisa já foi efectuada antes), de acesso (que documentos são retornados por cada pesquisa), de similaridade (quão similar é a pesquisa relativamente aos documentos do repositório) e de actualização (se um novo documento partilha conteúdos com outros documentos guardados). O racional por detrás destas soluções é que enquanto estes padrões não forem revelados (ou seja, enquanto não forem feitas pesquisas e/ou adicionados novos documentos), nada é revelado pelo índice ou pelo sistema (equivalente a segurança semântica [18]); depois de algumas operações, estes padrões vão sendo revelados, mas ainda assim o provedor na nuvem não consegue obter acesso aos dados em si.

Porém, esta combinação de padrões revelados implica que em sistemas CSP de longa vida, com muitas pesquisas efectuadas e vários documentos adicionados por múltiplos utilizadores, as garantias iniciais de segurança vão depressa ceder, degenerando para garantias de segurança deterministas (ou de preservação de similaridade). Garantias deterministas permitem que atacantes efectuem ataques estatísticos, apesar desse tipo de ataques estarem dependentes da disponibilidade de alguma informação de fundo sobre os dados cifrados [19]. Não obstante, as garantias iniciais de segurança são apenas possíveis com um ele-

vado custo computacional, que tornam sistemas CSP pouco práticos para dispositivos móveis de recursos limitados (devido a pesado processamento do lado do cliente [10, 14], múltiplas rondas de comunicação [13], e/ou elevado armazenamento no cliente [12]). Apesar de sistemas CSP poderem ser periodicamente refrescados através da re-cifra de todos os seus dados, repondo assim as garantias iniciais de segurança, tal operação teria de ser efectuada nos dispositivos dos clientes, o que mais uma vez revela a impraticabilidade destas soluções.

De forma geral, a maior parte dos trabalhos em CSP têm se focado no suporte de pesquisas Booleanas sobre documentos de texto [11–13]. Tais pesquisas, que normalmente apenas suportam uma palavra chave de cada vez, retornam todos os documentos que contêm essa palavra chave (correspondendo a centenas ou milhares em grande repositórios [12]), sem qualquer indicação de relevância. Alguns trabalhos da literatura suportam pesquisas de texto ordenadas por relevância, no entanto estes ou são baseados em primitivas criptográficas que já foram demonstradas serem inseguras [20, 21], or necessitam de pesado processamento pelo cliente e demonstram tempo de pesquisa linear com o tamanho do índice (que normalmente tem pelo menos um terço do tamanho do repositório) [11]. Para além disso, estas soluções não conseguem lidar facilmente com repositórios alterados frequentemente, pois as pontuações de relevância guardadas nos seus índices dependem de estatísticas gerais dos repositórios que têm que ser atualizadas com cada modificação nos mesmos, através de múltiplas rondas de comunicação.

Sistemas CSP têm também sido usados noutros domínios multimédia, como em imagens [10, 15]. No entanto, as limitações de eficiência existentes nos sistemas CSP para texto são ainda mais visíveis nestes domínios, pois processar e indexar dados multimédia ricos (como imagens, audio e video) é uma operação computacional bastante pesada. Não obstante, tanto quanto sabemos as contribuições existentes para suporte eficiente de pesquisas sobre dados cifrados limitam-se às enunciadas, e este trabalho é portanto a primeira tentativa de suportar, com garantias de privacidade, o armazenamento e pesquisa de dados contendo múltiplos conteúdos multimédia simultaneamente (i.e. dados multimodais).

3 Visão Geral do Sistema

3.1 Modelo de Sistema

Neste trabalho propomos um middleware distribuído, com dois componentes principais: um componente confiável no dispositivo dos utilizadores, que fornece uma API para as aplicações móveis; e um segundo componente não confiável, que opera na infra-estrutura do provedor da nuvem. O modelo de sistema

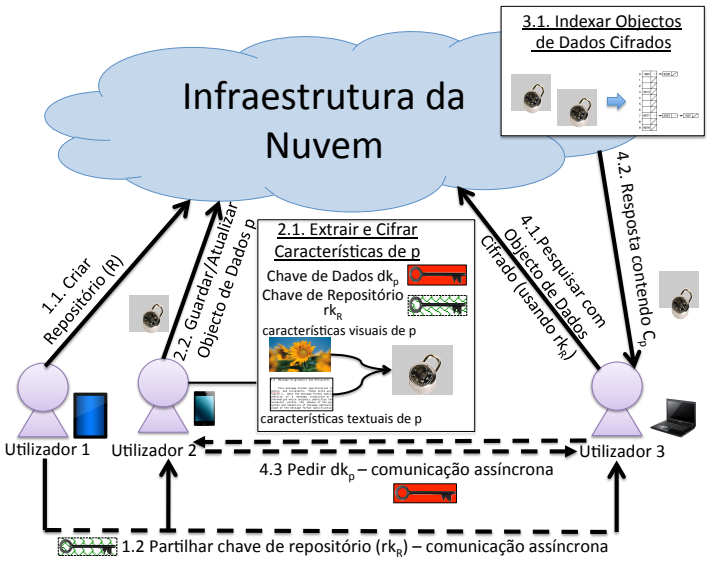


Figura 1: Modelo de Sistema

do trabalho proposto considera múltiplos leitores e escritores (a que chamamos **Utilizadores**), armazenando e partilhando dados multimodais através de múltiplos repositórios de dados armazenados num **Provedor da Nuvem** (ou simplesmente **Servidor da Nuvem**). Assumimos que todos os dados são enviados para estes repositórios, sob a forma de **Objectos de Dados** que podem conter múltiplos conteúdos multimédia. Cada repositório é criado por um utilizador, podendo depois ser usado por múltiplos outros utilizadores (confiáveis) que podem inserir os seus próprios objectos de dados multimodais, efectuar pesquisas multimodais, e obter objectos guardados no repositório. O objectivo do nosso middleware é assegurar a privacidade dos utilizadores, por isso todos os dados são cifrados antes de serem enviados para a nuvem. A Figura 1 mostra uma visão de alto nível do modelo de sistema descrito.

Após a criação de um repositório, o seu criador gera (e mais tarde partilha com os seus utilizadores confiados) uma **Chave de Repositório** (que pode ser composta por múltiplas chaves criptográficas, ex: uma para cada tipo de multimédia suportada pelo repositório). Esta chave fornece privilégios de pesquisa e de armazenamento para o respectivo repositório, e é usada na codificação CCD de características vectoriais dos objectos. Para além de chaves de repositório, o nosso modelo também emprega **Chaves de Dados**, que são usadas para cifrar os conteúdos dos objectos de dados[1]. Quando se pretende adicionar (ou actuali-

[1] Chaves de dados oferecem aos utilizadores um controlo mais fino sob quem acede aos conteúdos dos seus objectos; no entanto podem ser removidas do design do middleware se tal

zar) um objecto de dados num repositório, o utilizador deve primeiro processar o mesmo, extraindo as suas características vectoriais nas diferentes modalidades presentes e cifrando tanto o objecto como as suas características. Objectos de dados são cifrados usando esquemas de cifra convencionais (ex: AES em modo CBC) e chaves de dados geradas aleatoriamente; características vectoriais são cifradas usando os esquemas de CCD que propomos mais tarde neste artigo (secção 4), e a chave de repositório correspondente. O objecto de dados e suas características cifradas são então enviadas para o componente na nuvem do nosso middleware, que os armazena e procede à indexação do objecto através das suas características.

Os utilizadores também podem efectuar pesquisas multimodais, utilizando para tal novos objectos de dados com qualquer número de modalidades dentro das suportadas no repositório. Para tal, os utilizadores processam os objectos das suas pesquisas tal como processaram os objectos do repositório. No entanto apenas precisam de enviar para a nuvem as características vectoriais cifradas, podendo descartar o objecto de dados em si. Após receber uma destas pesquisas multimodais, o componente na nuvem do middleware encarrega-se de a processar e devolver as k entradas mais relevantes, onde k é um parâmetro configurável. Cada uma destas k entradas contém um par de objectos de dados cifrados e metadados para o objecto, incluindo o id do dono do objecto. Para aceder aos dados completos de um objecto, o utilizador que fez a pesquisa deve então pedir acesso à chave de dados desse objecto (a não ser que chaves de dados tenham sido removidas do modelo de sistema, como discutido previamente).

3.2 Modelo Adversarial

Neste trabalho procuramos proteger a privacidade dos dados dos utilizadores e das suas pesquisas. O adversário principal que consideramos é o *administrador da nuvem curioso*, que opera a infra-estrutura da nuvem e seus servidores. Semelhantemente a muitos trabalhos do estado da arte [10,11,13–15,20,21], assumimos um modelo de nuvem honesta mas curiosa [2], isto é, a nuvem é vista como um adversário passivo que deverá executar os seus protocolos correctamente quando pedido (i.e. cumprir com as suas obrigações contratuais), mas que ainda assim pode espiar em segredo sob os dados dos utilizadores. Assumimos que tal adversário tem acesso a todos os dados armazenados em memória principal ou secundária na infra-estrutura da nuvem, e a toda a informação que passa na rede. Neste trabalho não endereçamos ameaças de integridade e de disponibilidade, pois estas podem ser endereçadas por diferentes mecanismos

controlo não for necessário, usando para tal efeito, por exemplo, uma chave mestra partilhada por todos os utilizadores.

ortogonais aos escopo do artigo. Finalmente, assumimos que as aplicações que usam o nosso middleware conseguem controlar a quantidade de informação de fundo que revelam os seus dados, pois esta pode ser sensível e pode ser usada pelos adversários para quebrar a segurança do sistema.

4 Codificações Conservantes de Distância

Nesta secção propomos uma nova família de primitivas criptográficas, chamadas Codificações Conservantes de Distância (CCD), que serão usadas no desenho do middleware CMI. A nossa proposta advém da generalização e análise formal dos principais princípios por detrás de diferentes mecanismos existentes para computações de similaridade e de vizinhos-mais-próximos com garantias de segurança (e.g. [22, 23]). Começamos esta secção por definir formalmente o conceito de Codificação Conservante de Distância, e depois apresentamos duas instanciações do mesmo, uma aplicada a dados densos (ex: imagens) e outra aplicada a dados esparsos (ex: texto).

4.1 CCD Definição e Funcionalidade

Informalmente, definimos Codificações Conservantes de Distância (CCD) como uma família de esquemas criptográficos cujo algoritmo de codificação preserva uma função de distância controlável entre os dados em claro, através das respectivas codificações. Dizemos que a função de distância é controlável pois em instanciação de um esquema CCD, um limite de segurança é parametrizado, controlando assim a quantidade de informação revelada por esta função. Mais concretamente, esquemas CCD apenas devem preservar distâncias entre os dados em claro até ao limite de segurança. Para distâncias superiores, nada deve ser revelado pelas codificações CCD. A definição deste limite permite barrar a capacidade de um adversário em estabelecer uma relação de distância entre todos os dados do domínio de aplicação, que seria imprescindível na quebra de tais primitivas criptográficas. Mais formalmente:

Definition 1 (Codificação Conservante de Distância). *Um esquema de Codificação Conservante de Distância (CCD) é uma colecção de três algoritmos de tempo polinomial (*GERACHAVE, CODIFICA, DISTÂNCIA*), executados entre um utilizador e um servidor na nuvem, tal que:*

• $K, t \leftarrow$ GERACHAVE(1^k)*: é um algoritmo probabilístico de geração de chaves executado pelo utilizador. Leva como argumento o parâmetro de segurança k em unário e devolve a chave secreta K e o limite de segurança t, ambos função de e polinomialmente limitados por k.*

• $e \leftarrow$ CODIFICA(K, p)*: é um algoritmo determinístico executado pelo utilizador para codificar o dado em claro* p *com chave* K*, com* p *polinomialmente limitado por* k*. Devolve a codificação* e*.*

• $D \leftarrow$ DISTÂNCIA(e_1, e_2)*: é um algoritmo determinístico executado pelo servidor que leva como entrada duas codificações* e_1 *e* e_2*. Para duas funções distância* $[0, 1] \leftarrow d_p(\cdot, \cdot), d_e(\cdot, \cdot)$ *(possivelmente* $d_p = d_e$*) com entradas polinomialmente limitadas por* k*, o algoritmo devolve* $D = d_e(e_1, e_2) = d_p(p_1, p_2)$*, se e só se* $d_p(p_1, p_2) < t$*. Caso contrário devolve* $D = t$*.*

Notamos que a informação revelada por CCD é limitada, devido a t. Ainda assim um adversário pode usar tal informação para aprender algumas estatísticas sobre os dados e, com a ajuda de informação de fundo sobre os mesmos, possivelmente revelar os dados em claro. Como tal, as aplicações que usem CCD devem garantir que tal não acontece, limitando a informação de fundo revelada a adversários.

4.2 Uma Construção CCD para Dados Densos

Dados densos (ex: imagens, audio, video) são caracterizados por terem alta dimensionalidade e por terem sempre algum valor (diferente de zero) em todas as suas dimensões. Como tal, uma construção CCD para dados densos deve ser capaz de eficientemente codificar vectores de características de grande dimensionalidade, enquanto preserva uma função de distância controlável entre eles. Para tal, escolhemos estender e implementar (uma vez que esta não foi implementada pelos autores) a codificação de [22] para cálculo de vizinhos-mais-próximos com garantias de privacidade. Esta codificação dá garantias de segurança de informação teórica (*information-theoretic security*), enquanto preserva distâncias l_2 entre os dados em claro até um parâmetro configurável t, representadas pelas distâncias l_1 entre as codificações. Para distâncias superiores a t, a distância entre codificações tende para um valor constante. Mais concretamente, a codificação é dada pela seguinte formula:

$$e(x) = Q(\Delta^{-1}(Ax + w)) \tag{1}$$

onde $x \in \mathbb{R}^N$ é o vector de entrada com N dimensões, $A \in \mathbb{R}^{M \times N}$ é uma matriz aleatória com elementos independentemente e igualmente distribuídos no domínio designado (onde M é um parâmetro configurável que controla a expansão da codificação e o ruído introduzido pelo algoritmo), Δ é um parâmetro configurável que controla o limite de distância t, $w \in \mathbb{R}^M$ é um vector aditivo com valores uniformemente distribuídos em $[0, \Delta]$, e $Q(.)$ é uma função quantizadora que transforma valores num intervalo $[2v, 2v + 1[$ em 1 e valores entre

$[2v+1, 2v+2[$ em 0, para qualquer v. Neste esquema $\{A, w\}$ compõem a chave secreta.

O esquema anterior sofre de uma forte de limitação de aplicabilidade: a chave secreta $\{A, w\}$ deve ter tamanho proporcional ao tamanho das características vectoriais de entrada e às codificações resultantes (N e M respectivamente). Por um lado, esta propriedade do esquema leva a chaves criptográficas de grande dimensão; por outro lado, limita a flexibilidade do esquema pois uma mudança no tamanho do input/output (ex: mudança de características extraídas pelos utilizadores ou necessidade de reduzir o nível de ruído da codificação) obrigam à geração e partilha de novas chaves com o tamanho correcto. Para resolver esta situação introduzimos um Gerador Pseudo-Aleatório $\mathcal{G}$ [18] no algoritmo de geração de chaves, instanciado com uma semente criptográfica s contendo alguns bits de entropia. Os valores aleatórios de A e w são gerados a partir de $\mathcal{G}$, e assim os utilizadores apenas têm de partilhar entre si a semente criptográfica s (chegando sempre à mesma chave $\{A, w\}$, dados os parâmetros N e M). Para além do mais, dado um adversário probabilístico de tempo polinomial [18] os valores gerados por $\mathcal{G}$ serão indistinguíveis de valores verdadeiramente aleatórios [18]. Esta aproximação limita a segurança oferecida a *segurança computacional*, no entanto esta limitação não afecta a aplicabilidade do esquema pois neste trabalho assumimos, por razões obvias de praticabilidade, que o modelo adversarial descrito na secção 3.2 está restringido a tempo de execução polinomial. Chamamos à nossa construção *CCD-Densa*.

4.3 Uma Construção CCD para Dados Esparsos

Contrariamente a dados densos, dados esparsos são caracterizados por terem, de toda a sua dimensionalidade, um limitado número de valores não nulos. Um exemplo são dados de texto, onde um documento de texto terá apenas um subconjunto de todas as palavras possíveis dado o vocabulário da língua considerada. Assim sendo, é possível usar algoritmos de indexação e pesquisa mais eficientes para dados esparsos, comparativamente com dados densos. Mais concretamente, para indexar e pesquisar em dados esparsos, basta comparar para igualdade os diferentes valores não nulos constantes nas características vectoriais dos objectos de dados (ex: o histograma de palavras chave de cada documento de texto). Traduzindo estes requisitos para a definição CCD, a nossa construção CCD para dados esparsos terá um limite de distância fixo com valor $t = 0$, ou seja, apenas revelará se duas palavras chave são iguais ou não, e nada será revelado mesmo que tenham um byte de diferença (ex: um caracter diferente). A nossa construção baseia-se em Funções Pseudo-Aleatórias [18], mais especificamente:

$$e(x) = \mathcal{P}_K(x) \tag{2}$$

onde x é a característica de entrada (ex: uma palavra chave) e $\mathcal{P}$ é uma função pseudo-aleatória instanciada com chave secreta K (exemplo, SHA1-HMAC [18]). Chamamos à nossa construção *CCD-Esparsa*.

5 Desenho de Middleware CMI

Nesta secção descrevemos em detalhe a nossa proposta de um middleware para Cifra Multimodal Indexável, a que chamamos CMI. CMI é definido como uma arquitectura middleware distribuída, com dois componentes principais: um componente confiável de extracção de características vectoriais e de cifra, que executa nos dispositivos móveis dos utilizadores; e um componente não confiável de indexação e armazenamento, que corre nos servidores da nuvem. Estes componentes usam a nossa definição anterior de Codificações Conservantes de Distância (CCD, secção 4), assim como as duas construções propostas para dados densos e esparsos (CCD-Densa e CCD-Esparsa, respectivamente) para conseguir concretizar as suas funcionalidades. Mais concretamente, CMI é definido pela seguinte API:

Definition 2 (Cifra Multimodal Indexável). *Um middleware de Cifra Multimodal Indexável (CMI) é uma colecção de cinco operações executadas colaborativamente entre os dois componentes que executam nos dispositivos do utilizador e na infra-estrutura da nuvem, respectivamente, tal que:*

• $\{K_R^i, t_R^i\}_{i=0}^n \leftarrow$ CRIARREPOSITÓRIO$(ID_R, \{ID_{m_i}, sp_{m_i}\}_{i=0}^n)$*: é uma operação iniciada pelo utilizador para criar um novo repositório. Leva como entrada um id determinístico do repositório (ID_R), e ids e parâmetros de segurança das n modalidades suportadas ($\{ID_{m_i}, sp_{m_i}\}_{i=0}^n$). Gera uma nova chave criptográfica para cada modalidade suportada e cria uma representação do repositório na infra-estrutura da nuvem. O algoritmo devolve as chaves criadas, assim como os limites de distância para cada modalidade.*

• TREINAR$(ID_R, \{ID_{m_i}, ip_{m_i}\}_{i=0}^n)$*: operação invocada pelo utilizador e executada pela nuvem para inicializar as estruturas de indexação do repositório R (que pode incluir a execução de procedimentos de treino automatizados) e para indexar os seus atuais objectos de dados, se existirem. O algoritmo leva também como entrada alguns parâmetros de indexação ($\{ip_{m_i}\}_{i=0}^n$, um para cada modalidade). Este algoritmo pode ser invocado múltiplas vezes pelos utilizadores, com diferentes parâmetros de indexação, e é executado em plano de fundo pelo componente middleware na nuvem.*

• ACTUALIZAR$(ID_R, ID_p, p, dk_p, \{ID_{m_i}, K_{m_i}\}_{i=0}^n)$*: é a operação invocada para adicionar dinamicamente (ou actualizar, através da prévia execução do algoritmo* Remover*) o objecto de dados p no repositório R. Leva como entrada ID_R, ID_p, p, dk_p (chave de dados usada para cifrar o conteúdo de p),*

e $\{ID_{m_i}, K_{m_i}\}_{i=0}^{n}$ *(chaves de repositório de* R *para cada modalidade contida por* p*, que podem ser um subconjunto das de* R*). O utilizador extrai e cifra com CCD as características vectoriais de* p*, enviando para a nuvem. Se a operação de* Treinar *já foi invocada em* R*, o componente da nuvem indexa* p *através das suas características, caso contrário apenas guarda estas (e* p*) em memória secundária.*

• REMOVER(ID_R, ID_p)*: permite a um utilizador remover o objecto de dados* p *do repositório* R *(requer que o utilizador seja dono de* p*).*

• $\{ID_{p_i}, p_i, score_{p_i}^{q}\}_{i=0}^{k} \leftarrow$ PESQUISAR$(ID_R, q, \{ID_{m_i}, K_{m_i}\}_{i=0}^{*}, k)$*: é invocado pelo utilizador para pesquisar no repositório* R *com o objecto de dados* q*, retornando os* k *objectos de dados mais relevantes. Também leva como entrada as chaves de repositório para cada modalidade contida em* q *(*$\{K_{m_i}\}_{i=0}^{*}$*). O utilizador extrai as características vectoriais de* q *e cifra-as com CCD, enviando-as para a nuvem. Se a operação de* Treinar *já foi invocada em* R*, o componente na nuvem responde à pesquisa em tempo sub-linear acedendo às estruturas de indexação de* R*. Caso contrário, responde efectuando um varrimento linear sob os objectos de dados guardados em* R*.*

Devido a limitações de espaço, omitimos uma descrição mais detalhada e algorítmica dos protocolos do middleware CMI.

6 Implementação e Avaliação de CMI

Nesta secção descrevemos uma possível implementação do middleware CMI proposto na secção 5. A partir desta implementação de referência foi criado um protótipo para validação experimental das propriedades do CMI, em termos de desempenho, escalabilidade e impacto de bateria nos dispositivos móveis dos utilizadores. Para comparação, implementámos também um protótipo CSP de referência, inspirado pela literatura do estado da arte [10–15] (foi necessário implementar de raiz um protótipo CSP pois nenhum dos trabalhos do estado da arte oferece uma implementação de referência e muitos são apenas teóricos). Ambos os protótipos foram implementados em C++ e Android NDK[2]. Uma pequena aplicação Android foi também implementada, de forma a explorar a API de ambos os protótipos. Como demonstração de multimodalidade, ambos os protótipos foram desenvolvidos para suportar os domínios de texto e imagens. As bibliotecas OpenSSl 1.0.2[3] and OpenCV 2.4.10[4] foram usadas para suportar operações criptográficas e de processamento de imagens, respectivamente.

[2] http://developer.android.com/
[3] https://www.openssl.org/
[4] http://opencv.org/

A extracção de características vectoriais de texto consistiu em *stemming* de palavras chave, remoção de *stop-words* e extracção de histogramas [24]. A extracção em imagens foi feita por detecção de pontos chave *SURF* [25] e extracção de descritores *Dense Pyramid* [26]. Foram usados índices de lista invertida em ambos os domínios [24], e o modelo *Bag-Of-Visual-Words* [27] para representar os descritores de imagens como palavras visuais (*visual words*), com uma árvore de vocabulário visual de 1000 nós folha (altura 3 e ramificação 10) obtidos através de clusterização *k-means* na operação *Treinar*. A função de pontuação BM25L [28] foi usada no domínio de texto e TF-IDF [27] para pontuação de imagens. *ISR Rank Fusion* [29] foi usado para fundir os resultados de pesquisa nos diferente domínios multimédia. Mais uma vez relembramos que esta é apenas um das múltiplas combinações possíveis, dada a flexibilidade do nosso middleware em incorporar diferentes multimédia e diferentes técnicas de indexação e de pesquisa de informação.

De seguida apresentamos os resultados experimentais obtidos. Estes foram conseguidos usando uma tablet Nexus 7 (Android 5.1.0) como dispositivo móvel dos utilizadores e um Macbook Pro (MAC OS X 10.10.3, 4GB de Ram e 2.3Ghz Core i7 CPU) como servidor na nuvem. A comunicação entre ambos foi feita através de uma rede local, onde a tablet Android comunicava por sinal WIFI. Nas experiências que descrevemos de seguida, foi usado o conjunto de dados 1M MIR-Flickr [30], que contem um milhão de imagens, e suas anotações, retiradas da rede social Flickr. A nossa avaliação experimental focou-se no desempenho (em tempo gasto pelos dispositivos móveis) e escalabilidade (impacto do tamanho do conjunto de dados e número de utilizadores a interagirem com ele) para a operação de *Actualização*. Adicionalmente, também analisámos o impacto de bateria nos dispositivos móveis desta operação.

6.1 Desempenho e Escalabilidade

Para avaliar experimentalmente o desempenho e escalabilidade do nosso middleware relativamente à operação de *Actualização*, efectuámos duas experiências: na primeira experiência, um utilizador cria um novo repositório e adiciona 1000 objectos de dados (imagens e respectivas anotações) escolhidos aleatoriamente do conjunto de dados MIR-FLICKR; na segunda experiência, depois do primeiro utilizador iniciar o repositório, mais 100 utilizadores (simulados por processos leves no dispositivo móvel) adicionam simultaneamente 10 novos objectos de dados cada um (total de mais 1000 objectos). A Figura 2 resume os resultados obtidos, com uma média de 5 execuções independentes. Para cada experiência medimos: i) tempo de cifra de dados e características vectoriais (*Cifra* na figura); ii) tempo de transmissão de dados pela rede (*Nuvem* na figura); iii) tempo gasto em computações de indexação pelo dispositivo móvel (*Indexação*);

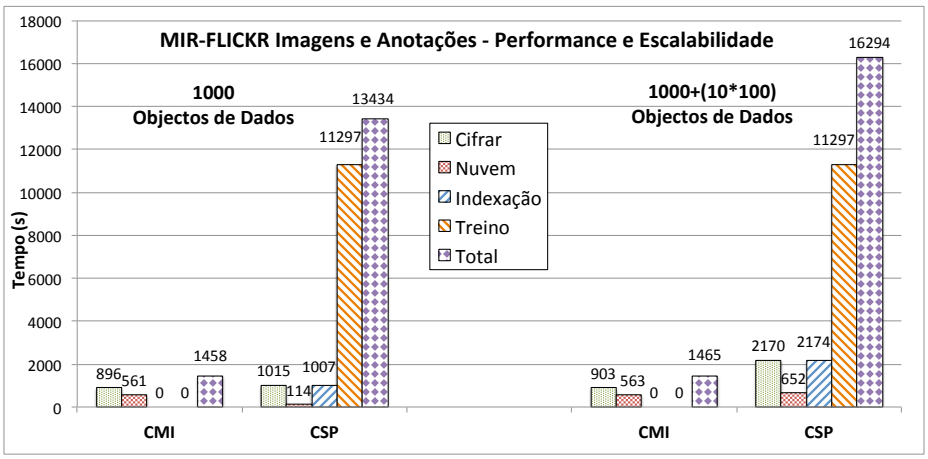


Figura 2: Desempenho e Escalabilidade da operação de Actualização

e iv) tempo gasto na operação de iniciação do repositório (*Treino*). Para conveniência, também incluímos o tempo total de ambas as experiências (*Total*).

Os resultados da primeira experiência (parte esquerda da Figura 2) mostram que na aproximação CSP, a operação de treino tem um grande impacto em termos de tempo gasto pelo utilizador. Isto é devido à necessidade de executar, neste passo, tarefas de clusterização e aprendizagem máquina, especialmente sobre tipos de dados densos (as imagens). Na nossa experiência apenas 10% dos 1000 objectos de dados foram usados como conjunto de dados de treino, o que significa que na realidade os resultados demonstrados representam apenas uma pequena fracção do impacto que este passo pode introduzir. Em contraste claro, o nosso middleware CMI permite que estas computações de aprendizagem máquina sejam executadas na nuvem com garantias de privacidade, evitando assim o seu impacto computacional nos dispositivos móveis. Para além do mais, quando efectuada pela infra-estrutura na nuvem, a operação de treino pode ser executada em plano de fundo (ao contrário de na aproximação CSP), o que contribui ainda mais para a escalabilidade da nossa proposta.

Não obstante, o leitor atento pode argumentar que a operação de treino só tem estritamente de ser executada uma vez por repositório, o que significa que o seu custo amortizará com tempo. No entanto o impacto das restantes operações demonstra que tal não é o caso, pois mesmo nestas a nossa aproximação CMI continua a ter melhor desempenho que CSP, com uma diferença de 31% (1458 segundos no CMI comparado com 2136 segundos em CSP, se retirarmos o tempo de treino). Para além disso, a segunda experiência (parte direita da Figura 2) mostra que a nossa aproximação também mostra melhores propriedades de escalabilidade se considerarmos múltiplos utilizadores. Ao introduzir-

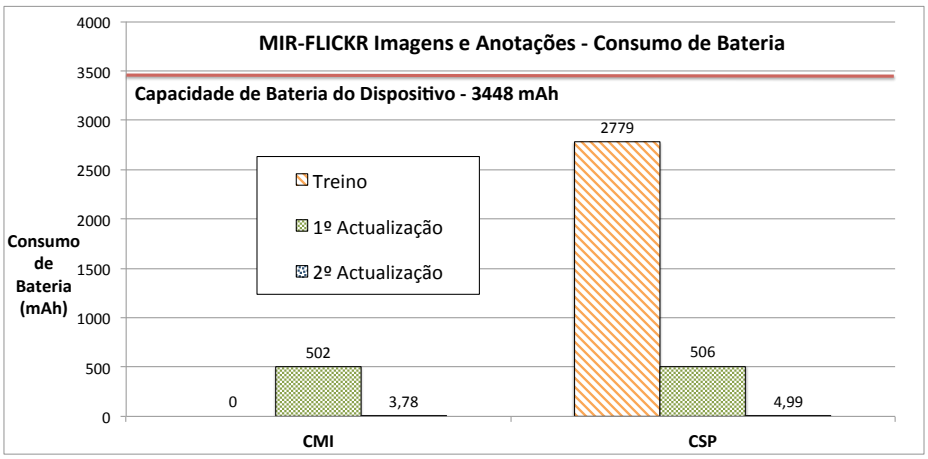


Figura 3: Consumo de Bateria para as experiências de Actualização e operação de Treino

mos 100 utilizadores a adicionarem 10 novos objectos de dados cada simultaneamente, o tempo necessário pela aproximação CSP, em relação à experiência anterior, aumenta em 57%, enquanto em CMI verifica-se um aumento negligenciável de 0.5% (não considerando o tempo de treino, cuja operação não volta a ser executada nesta segunda experiência). Esta degradação de desempenho é explicada pela necessidade de, em CSP, os utilizadores terem que sincronizar entre eles como actualizar os índices do repositório, uma vez que tal operação só pode ser feita de forma segura nos seus dispositivos. Ao permitir que tal sincronização seja gerida pelo componente na nuvem de forma segura, CMI evita tais encargos e melhora o desempenho e escalabilidade do sistema.

6.2 Impacto em Recurso Móveis

Para avaliar o impacto no recursos móveis do nosso middleware CMI, comparativamente com CSP, voltámos a executar as duas experiências anteriores e medimos o uso de bateria por dispositivo móvel para as duas experiências. A Figura 3 mostra os resultados em microamperes/hora, onde *Treino* representa o impacto da operação de treino, *1º Actualização* representa o impacto da 1º experiência se retirarmos a operação de treino (ou seja, o impacto de adicionar 1000 objectos de dados), e *2º Actualização* representa o impacto da segunda experiência (i.e. adicionar 10 novos objectos ao repositório).

Como se pode constatar pela Figura 3, a operação de treino tem um grande impacto de bateria na aproximação CSP, consumindo à volta de 80% da capacidade de bateria do Nexus 7. Se a isto juntarmos o consumo de 14% de bateria da inserção dos 1000 objectos de dados iniciais (*1º Actualização* na figura), obtemos um total de 94% de bateria consumida para criar um novo repositório e

adicionar um conjunto de dados inicial. Em claro contraste, apesar de o nosso middleware CMI gastar aproximadamente a mesma bateria para adicionar os documentos, ele não requer que a operação de treino seja executada nos dispositivos móveis, evitando assim o seu elevado custo. Para além disso, na segunda experiência (*2º Actualização* na figura) cada dispositivo móvel caba por consumir 4.99 mAh em CSP, comparativamente com 3.78 mAh em CMI, o que mais uma vez demonstra a escalabilidade da nossa solução e a sua capacidade de suportar dispositivos móveis, ao contrário de soluções CSP da literatura.

7 Conclusões

Neste artigo propusemos uma nova arquitectura middleware distribuída, a que chamámos CMI, que permite que aplicações móveis guardem os seus dados multimodais em repositórios na nuvem, possibilitando ainda a sua pesquisa de forma eficiente. CMI preserva a privacidade dos utilizadores contra operadores da nuvem honestos-mas-curiosos e outros espiões na rede, tanto relativamente aos dados armazenados na nuvem como quanto às pesquisas efectuadas sobre os mesmos. No centro do CMI encontra-se uma nova família de algoritmos criptográficos, chamada Codificações Conservantes de Distância (CCD), que formalmente definimos e analisámos, resultando na proposta de duas instanciações diferentes: uma para tipos de dados densos (ex: imagens), e outra para tipos de dados esparsos (ex: texto). Foi implementado um protótipo do middleware distribuído CMI, assim como uma aplicação Android capaz de operar sobre dados de texto e imagens, e foi experimentalmente demonstrada a vantagem da nossa aproximação quando comparada com o estado da arte. Como trabalho futuro pretendemos avaliar a performance das operações de pesquisa, analisar e comprovar formalmente as propriedades de segurança das soluções propostas, e validar o design do nosso middleware com outros tipos de mídias (ex: audio e video) e outras plataformas móveis (ex: iOS).

Agradecimentos: Este trabalho foi parcialmente suportado pela FCT através dos projectos PEst UID/CEC/04516/2013 e CMUP-ERI/FIA/0048/2013.

Referências

1. Meeker, M.: Internet Trends. In: Code Conf. (2014) 1–164
2. Chow, R., Golle, P., Jakobsson, M., et al.: Controlling data in the cloud: outsourcing computation without outsourcing control. In: CCSW'09. (2009)
3. Rushe, D.: Google: don't expect privacy when sending to Gmail. http://tinyurl.com/kjga34x (2013)
4. Greenwald, G., MacAskill, E.: NSA Prism program taps in to user data of Apple, Google and others. http://tinyurl.com/oea3g8t (2013)

5. Lewis, D.: iCloud Data Breach: Hacking And Celebrity Photos. https://tinyurl.com/nohznmr (2014)
6. Chen, A.: GCreep: Google Engineer Stalked Teens, Spied on Chats. http://gawker.com/5637234 (2010)
7. Frieden, T.: VA will pay $20 million to settle lawsuit over stolen laptop's data. http://tinyurl.com/lg4os9m (2009)
8. Halderman, J., Schoen, S.: Lest we remember: cold-boot attacks on encryption keys. In: Commun. ACM. Volume 52. (2009) 91–98
9. Mahajan, P., Setty, S., Lee, S., Clement, A., Alvisi, L., Dahlin, M., Walfish, M.: Depot: Cloud Storage with Minimal Trust. ACM Trans. Comput. Syst. **29**(4) (December 2011) 1–38
10. Lu, W., Swaminathan, A., Varna, A.L., Wu, M.: Enabling Search over Encrypted Multimedia Databases. In: IS&T/SPIE Electron. Imaging. (February 2009) 725418–725418–11
11. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data. IEEE Trans. Parallel Distrib. Syst. **25**(1) (January 2014)
12. Cash, D., Jaeger, J., Jarecki, S., et al.: Dynamic searchable encryption in very-large databases: Data structures and implementation. In: NDSS'14. Volume 14. (2014)
13. Naveed, M., Prabhakaran, M., Gunter, C.A.: Dynamic Searchable Encryption via Blind Storage. In: IEEE S&P. Number May (2014)
14. Hahn, F., Kerschbaum, F.: Searchable Encryption with Secure and Efficient Updates. In: CCS'14, ACM (2014) 310–320
15. Yuan, X., Wang, X., Wang, C., Squicciarini, A., Ren, K.: Enabling Privacy-preserving Image-centric Social Discovery. In: ICDCS'14, IEEE (2014) 198–207
16. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: CRYPTO'12. Springer (2012) 850–867
17. Stefanov, E., Dijk, M.V., Shi, E., Fletcher, C., Ren, L., Yu, X., Devadas, S.: Path oram: An extremely simple oblivious ram protocol. CCS 2013 (2013) 299–310
18. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. CRC PRESS (2007)
19. Islam, M.S., Kuzu, M., Kantarcioglu, M.: Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In: NDSS. (2012)
20. Wang, C., Cao, N., Ren, K., Lou, W.: Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data. IEEE Trans. Parallel Distrib. Syst. **23**(8) (2012) 1467–1479
21. Popa, R.A., Li, F.H., Zeldovich, N.: An Ideal-Security Protocol for Order-Preserving Encoding. IEEE S&P (May 2013) 463–477
22. Boufounos, P., Rane, S.: Secure binary embeddings for privacy preserving nearest neighbors. In: IEEE Int. Work. Inf. Forensics Secur., Ieee (November 2011) 1–6
23. Ferreira, B., Domingos, H.: Searching private data in a cloud encrypted domain. In: Proc. 10th Conf. Open Res. Areas Inf. Retr. (OAIR '13), C.I.D. (May 2013) 165–172
24. Manning, C.D., Raghavan, P., Schütze, H.: An Introduction to Information Retrieval. Volume 1. Cambridge University Press (2009)
25. Bay, H., Tuytelaars, T., Gool, L.V.: SURF: Speeded Up Robust Features. In: Comput. Vision-ECCV 2006, Springer (2006) 404–417
26. Lazebnik, S., Schmid, C., Ponce, J.: Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In: CVPR'06. Volume 2., IEEE (2006) 2169–2178
27. Nister, D., Stewenius, H.: Scalable recognition with a vocabulary tree. In: CVPR. Volume 2., IEEE (2006) 2161–2168
28. Lv, Y., Zhai, C.: When documents are very long, BM25 fails! In: ACM SIGIR. Number I, New York, New York, USA, ACM Press (2011) 1103–1104
29. Mourão, A., Martins, F., Magalhães, J.a.: NovaSearch at TREC 2013 Federated Web Search Track : Experiments with rank fusion. In: TREC. Number Task 1 (2013) 1–8
30. Huiskes, M.J., Lew, M.S.: The MIR Flickr Retrieval Evaluation. In: MIR '08, New York, NY, USA, ACM (2008)

On the Susceptibility to Data Manipulation and Information Exposure of Free Android Apps with In-app Purchases

Francisco Vigário[1], Miguel Neto[1], Musa G. Samaila[1,2], Mário M. Freire[1] and Pedro R. M. Inácio[1]

[1] Instituto de Telecomunicações and Department of Computer Science, University of Beira Interior, *Rua Marquês d'Ávila e Bolama, 6201-001 Covilhã, Portugal*
[2] *Centre for Geodesy and Geodynamics, National Space Research and Development Agency, Toro, Bauchi State, Nigeria*
fvigario@penhas.di.ubi.pt, {miguel.neto, mgsamaila}@it.ubi.pt, {inacio,mario}@di.ubi.pt

Abstract. Data stored by Applications (apps) in mobile devices constitutes a tempting target for attackers, either due to the information it refers to, or because it may be used to control their flow and functionalities. This paper describes a detailed experimental study of the effects of storing sensitive data like credentials on a device; it also assesses the number of free Android apps with *in-app purchases* that are susceptible to data manipulation. To achieve these objectives, 693 apps were downloaded from Google Play store, and after using some criteria to filter the downloaded apps, 377 were manually analyzed for the purpose of this study. These apps were first used on a smartphone, then transfered to a computer with Linux Operating System using the Android backup utility. The entire process uses only freely and readily available tools. Results show that approximately 20% of the apps do not include native mechanisms to detect or prevent the data from being eavesdropped, or manipulated to change their execution flow or functionalities.

1 Introduction

The growth in smartphone usage has given rise to a significant growth in the number of companies [10, 15] developing mobile Operating Systems (OSs) that meet the increasing user needs for convenience. The major smartphone OSs, including Android, Apple and Windows, usually have App Stores that provide users with point and click access to thousands of both paid and free apps. The app market consisting of Apple's App Store, Windows' Phone Store and Google's Play Store has grown significantly in the last few years. Most importantly, Google Play Store is taking the major share of 75% of the total market-share [2]. This growth is underpinned by the massive adoption of smarthphones and the ubiquity of low cost apps on different mobile operating systems [8], resulting from competition among developers.

Modern mobile app stores are seen as business opportunity for developers of mobile apps. There are several business strategies used by developers. For example, a number of developers choose to ask for payment during the installation process, while others prefer to provide their apps for free and offer the possibility to purchase new features or get rid of publicity after the installation process, which is known as in-app purchasing [4]. Sometimes, the features to be purchased in apps have already been implemented and included, but are blocked using some programming logic that make them unusable, so as to prevent users from accessing them before payment. Typically, this programming logic is based in the values of variables that are stored as app information in the internal storage of the device. In addition to these variables, many developers also use the internal memory to store sensitive information related to the users (e.g., login credentials). In such cases, developers assume that the internal storage is protected (e.g., by the OS) against other apps and also that direct acess from users is difficult. Such assumptions raise security and privacy concerns, especially considering the fact that very sensitive personal data may be flowing over open heterogeneous networks in an insecure manner. Malicious users or interested developers can exploit these security flaws using the method described in this paper or other type of attacks and steal or manipulate important information for their own profit.

In this paper we exploit a method for accessing and manipulating the internal data storage of Android apps with *in-app purchases* using the backup utility provided by the Android OS without requiring root priviledges and readily available tools. In order to perform this analysis we define a data set that is composed of more than 690 apps downloaded from *Google Play* store. Nonetheless, some of those apps do not meet the criteria for the analysis and the number of apps considered for the final analysis is 377. These 377 apps were first installed on a non-rooted device, and then transfered to a Personal Computer (PC), where they were subjected to human analysis using common native text manipulation tools. The analysis process involves backing up the apps on a PC, searching for any data of interest, replacing it, re-installing the application on the smarthphone and observing their behaviour, in order to see whether the behaviour of the application is affected as a result of tampering with it. Results show that a significant percentage of the analyzed apps are susceptible to data manipulation. The results also reveal that some developers do not follow security best practices when storing information on smartphones internal memory. As such, curious users using preprepared targeted scripts, or users with advanced technical knowledge can enjoy paid Services without payment.

The remainder of this paper is structured as follows. Section 2 presents some related works and the motivation for this work. Section 3.1 describes the data set used in scope of this study and elaborates on the type of apps considered. This section also describes the method used to perform the analysis. The final results, as well as the number and type of apps that can be manipulated using the described method are discussed in section 4. Section 5 contains the main conclusions and presents some lines to be considered in the future.

2 Related Work

With the increased popularity of the Android OS and the unprecedented adoption of smartphones by people worldwide, an enormous number of mobile apps have been developed, most of them for Android. As a result, attackers are beginning to focus more on Android apps, posing serious threats to the stakeholders. In order to reduce the undesirable effects on users and help them to have some control over the apps they install, there are Internet platforms [3, 6, 12, 14] that conduct static analysis of files that require users to submit a file with `.apk` extension, which in turn outputs a report that describes the malware it contains. However, this has not solved the problem, and researchers are still trying to address the malware issue.

For example, Knorr and Aspinall [7] proposed a testing method for Android mHealth apps they designed, by employing a threat analysis using possible attack scenarios and vulnerabilities specific to the area, rather than out-of-the-box Android security testing methods. The authors applied the proposed method on apps for managing hypertension and diabetes in order to demonstrate its effectiveness, and the results they obtained show that there are many serious vulnerabilities in most popular apps.

In their quest to analyze the vulnerabilities of apps, Shuai et al. [11] proposed a vulnerability analyzing method that combines taint analysis and cryptography misuse detection. Their method consists of four essential steps, which are: decompile, taint analysis, Application Programming Interface (API) call record and cryptography misuse analysis. As a proof of concept, they developed a prototype tool called PWExam that can be used to analyze the way passwords are stored and handled in apps. The authors discovered that 4 out of the 12 apps they analyzed leaked the passwords of the users, hence they were vulnerable.

Since 2011, the Open Web Application Security Project (OWASP) [9] has been working on a project focused on threats in mobile devices. The report of this nonprofit organization on vulnerabilities, released at the end of 2014, shows that Insecure Data Storage ranked second, showing the level of severity in mobile apps.

A study by Palo Alto Networks and led by researcher C. Xiao, resulted in a presentation with the title *Insecure Internal Storage in Android*, which was presented in the Taiwan Conference (HITCON) [16]. This presentation elaborates on the same subject of this work. In this stud, the author analyzed 12,351 apps available in *Google Play*, and found that only 556 of these did not allow backup using the native backup utility available in Android. I.e., approximately 94% of the most popular apps allow the transfer of the package and all the internal storage files to the computer using the utility. The application developer can allow or disallow the backup of the application to the computer, by adjusting the property `android:allowBackup` to `true` or `false` in the `AndroidManifest.xml` .

In his dissertation, titled *Android Mobile Application Security with OWASP Top 10 2014*, King [5] conducted an analysis with the `FourGoats` Android app, where he identified several types of vulnerabilities, including the types mentioned

in this work (i.e., *Insecure Data Storage*). It was an indication that many apps store user credentials in plaintext using a local database in the external memory device.

In the same line of research, Haland [1], in his dissertation titled *An Application Security Assessment of Popular Free Android Applications*, conducted a study of the 20 most popular apps from Google Play and tested the OWASP Top 10 risks. He found several vulnerabilities in the studied apps such as *Insecure Data Storage*, *Weak Server Side Controls*, *Insufficient Transport Layer Protection* among others. For example, the author states that, in the *4Pics1Word* game, one can increase the number of virtual currency without playing for it or buy coins with real money. However, in this case, the device needed to be rooted to carry out such attacks.

3 Data Set and Method

This section presents the data set of apps used in the study reported herein and briefly presents the method to perform it.

3.1 Data Set

This study focuses on Android apps available in *Google Play*. For this study, a data set consisting of a total of 693 apps was created. Similarly to the data set in [13], all apps have a common characteristic, namely the presence of the *in-app purchase* possibility.

Table 1 shows the distribution of the apps through their categories, in accordance with the *Google play* store classification. Notice that we have set the maximum amount of apps per category to 30, to keep this study within feasible limits for the available time frame. The categories with less than 30 apps were the ones for which we could not find more with the desired feature of *in-app purchase*. This was our feature of interest because it defines apps that we can freely download the app and were data manipulation will probably have financial impact. A main concern in the construction of the data set is trying not to focus only on apps with greater popularity. Table 2 shows that all download intervals were considered, resulting in a large degree of heterogeneity in the data set.

3.2 Method

The method employed in this study consists of three essential steps, namely: (i) Transfer, (ii) Modification and (iii) Restore of the app into, and from, a desktop or laptop computer. The analysis was performed in a Linux environment, where all the necessary tools are already installed, with the exception of the Android Debug Bridge (ADB) tool, which can be easily installed before implementing the method. Each of the steps are described in greater detail below:

Table 1. Number of analyzed apps per category.

Category	Number of Games
Books & Reference	30
Business	30
Comics	30
Communication	30
Education	30
Entertainment	30
Finance	30
Health & Fitness	30
Libraries & Demo	11
Lifestyle	30
Media & Video	30
Medical	30
Music & Audio	30
News & Magazines	30
Personalization	30
Photography	30
Productivity	30
Shopping	23
Social	30
Sports	30
Tools	29
Transportation	30
Travel & Local	30
Weather	30

Table 2. Number of apps in the data set per number of downloads.

Number of Downloads	Number of apps per Interval
100M - 500M	4
50M - 100M	9
10M - 50M	42
5M - 10M	40
1M - 5M	120
500k - 1M	71
100k - 500k	197
50k - 100k	61
10k - 50k	89
5k - 10k	17
1k - 5k	23
500 - 1k	6
100 - 500	11
50 - 100	2
10 - 50	1

1) *Transfer* – The first step is to connect the smartphone to the computer using USB. The OS needs to be with the *debug mode* activated. The application to be modified can then backed up to the computer using the `adb` command by triggering the backup utility. The next stage in this step involves removing the header and converting the backed up file to the `.tar` file type. This can be accomplished with Linux common tools such as `openssl` and `tar`. Before unpacking the file, it is important to keep an accurate list of the files and directory structure, so that it can be easy to accurately reproduce the original file after the manipulation process.
2) *Modification* – The second step in this method consists of modifying the application data. This involves trying to tamper with the application data so as to have access to paid functionalities for free, or change the behaviour of the application. It also involves assessing if confidencial or private data (e.g., passwords or Personal Identification Numbers (PINs)) are stored safely or not. Pattern search and text manipulation tools such as `grep`, `find` or `nano` were used in this step. `SQLite` was also used to manipulate database files, when applicable.
3) *Restore* – The last step consists of inserting the header that was previously removed, packaging the modified app, and re-installing it back on the smartphone. After the restoration process, the application can then be tried again in order to verify the effectiveness of this method on the smartphone environment.

This method is described thoroughly in [13]. The backup utility is a tool of the Android OS itself, readily available to any user. The problem is that it exports application data that is protected in the OS environment, but not on a foreign system. With the right tools, one can easily access and manipulate the backed up data.

4 Discussion and Results

In contrast to the study in [13], which exclusively focused on the susceptibility to data manipulation of Android games with *in-app purchases*, here we broaden the scope to the other categories of apps (not only games) and extended the assessment to estimate the data exposure in terms of private information. I.e., in [13], we only assessed susceptibility to data manipulation. Herein, we study data exposure too. The data set used for this study consists of 693 apps that are used in everyday life from all the 24 categories provided on *Google Play*. However, not all the 693 apps were used for the final analysis, because we used some criteria to filter out every application that does not contain the required information. The filtering criteria was:

1) Some of the apps were withdrawn by their developers, and thus they are no longer in the *Google Play* store;
2) There are apps that though listed as *in-app purchase* apps, do not effectively have the *in-app purchase* functionality except perhaps to deactivate advertisements.

3) The `android:allowBackup` property on some of the apps is set to `false` in the `AndroidManifest.xml`, meaning that they cannot be backed up on a laptop or any desktop computer.
4) Finally, the `BQ Aquaris E5 FHD` smartphone used for this work was not compatible with some of the apps.

After applying the aforementioned filtering criteria, 316 apps were removed leaving 377 apps, which correspond to about 54% of the initial data set. A large percentage of the apps in the initial data set were not analyzed because of the conditions described above. Nonetheless, the resulting data set is representative of all categories, as the criteria resulted in cuts on every category, as shown in Figure 1. The reason for applying the criteria to the larger data set at the end was mostly due to the structuring of the research. We started by building a script for downloading all apps with *in-app purchases*, to only analyse each app in a later stage.

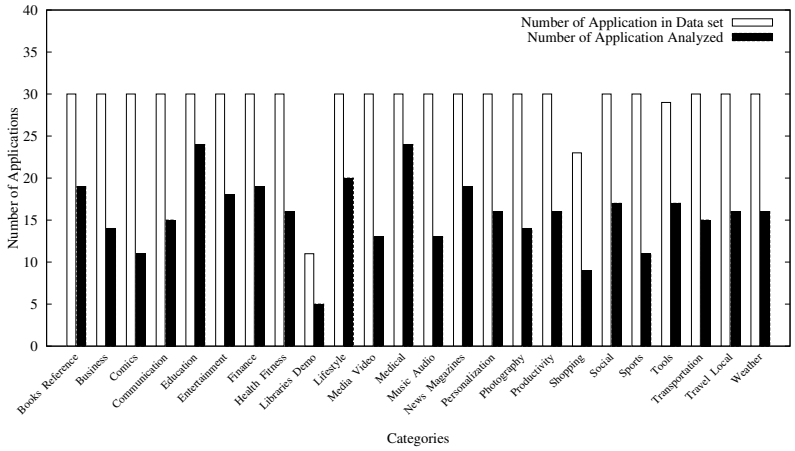


Fig. 1. Number of downloaded apps vs. number of apps fulfilling the conditions to analysis, divided per category.

Regarding the categories of apps listed in Table 1 that meet the criteria, the results in Figure 1 show that the *Medical* and *Education* categories are the categories that have more apps to be analyzed, with both having 80% of apps to be considered in the final analysis. This could be attributed greatly to the fact that the apps in these categories are more complex and have some paid add-ons that do not involve removal of advertisements. In the opposite side, one finds the categories of *Comics* and *Sports*, where only about 37% were considered for the final analysis. In the case of the Comics category, this can be explained by the fact that users only download the Comics for reading free or even read within the apps, and developers only use advertisements on the apps as a form of income. Users who prefer not to have advertisements in the apps can pay for

their removal. In the case of apps in the Sports category, which are mostly for showing soccer games and sports news, developers also use in-app advertisements for the same reason.

Still based on the previous relationship (i.e., between the number of apps in the initial data set and the number of apps that were finally selected for analysis), we also try to establish whether there is a relationship between the popularity of an app and the usage that developers make of the *in-app purchase* functionality. As shown in Figure 2, for apps with downloads in the ranges 50M to 100M and

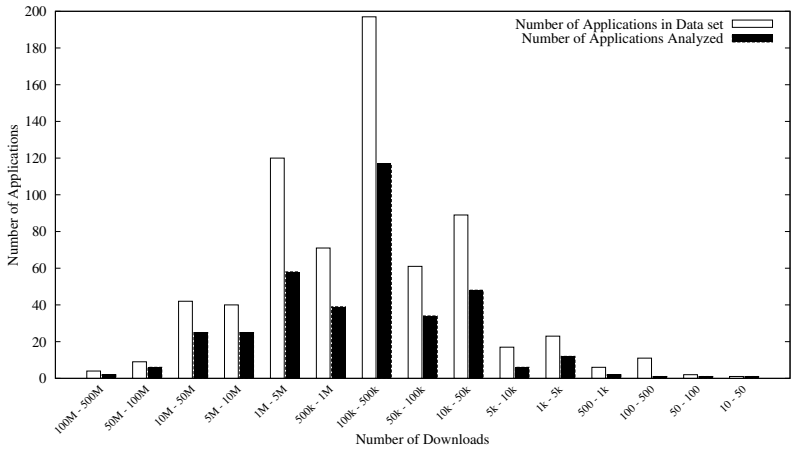


Fig. 2. Number of downloaded apps vs. number of apps fulfilling the conditions to analysis, divided per popularity.

5M to 10M, the percentage considered for the final analysis are 67% and 62% respectively. This shows that there is some relationship between the popularity of an application and the usage of the *in-app purchases* functionality. The apps that were excluded from these ranges were actually the ones providing a more limited set of features (they were generally simpler). On the other hand, the intervals 500 to 1k and 100 to 500 have the lowest percentage of apps considered, corresponding to approximately 33% and 9%, respectively, which corroborates the earlier assumption also.

Concerning the assessment of the apps susceptibility to data manipulation using the method described in Section 3.2, the analysis was divided into two phases. In the first phase, the number of apps that were found to be susceptible to data exposure or manipulation are compared with the total number of apps in the data set, for each one of the Google Play categories. In the second phase, we considered the relationship between popularity of apps and their susceptibility to data exposure or manipulation.

The results depicted in Figure 3 show that the *Media & Video* category is the one with the larger number of affected apps, with 46% of the apps susceptible to

data exposure or manipulation, corresponding to six vulnerable apps out of the thirteen considered for this study. This can be attributed to the fact that apps in this category, especially video apps, have tools that enable users to perform editing of recorded videos. It is common for developers, however, to place a number of mechanisms blocking video editing, which can be unlocked for a fee, thereby serving as the source of income for the developers. The Finance category follows next with 42% of vulnerable apps, which corresponds to eight out of the nineteen apps considered. This is due to the fact that most apps in this category allow logging into the apps, and the developers did not provide mechanisms for protecting user credentials, which are usually stored in plaintext in the auxiliary files. Another reason is that apps in this category usually have PIN for locking an application or some of its features, but unfortunately this code is stored in plaintext with no mechanisms to detect or prevent it from being modified or seen.

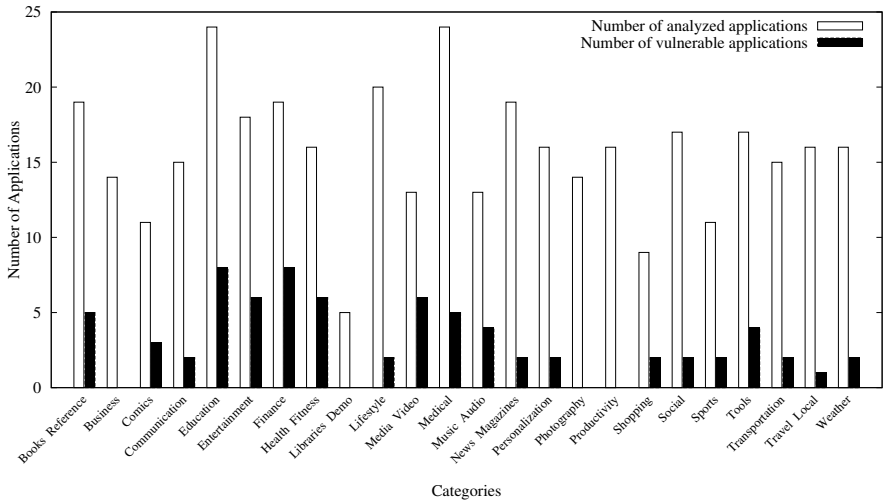


Fig. 3. Number of apps susceptible to data exposure or manipulation vs. number of analyzed apps, divided by category.

It is interesting to note that four categories of *Google Play* apps, namely Business, Libraries & Demo, Photography and Productivity had no vulnerable apps. This result can probably be explained by the fact that the *in-app purchases* in these apps are only used to remove advertisements, or the apps are purchased once and for all, and expanding or purchasing new levels of the apps requires downloading entirely new files, and hence giving no room for data manipulation.

Figure 4 shows the relationship between popularity of apps and their susceptibility to data manipulation. The figure also depicts that the apps whose number of *Google Play* downloads is in the range 50M to 100M are the most vulnerable, with about 33% of them being vulnerable. The result of this study

implies that the number of vulnerable apps is not dependent on their popularity. The fact that no vulnerable apps were found in the download interval 100M to 500M (which is one of five download intervals that have no vulnerable apps) does not suggest that popular apps are less vulnerable, since there were only 4 apps in this interval.

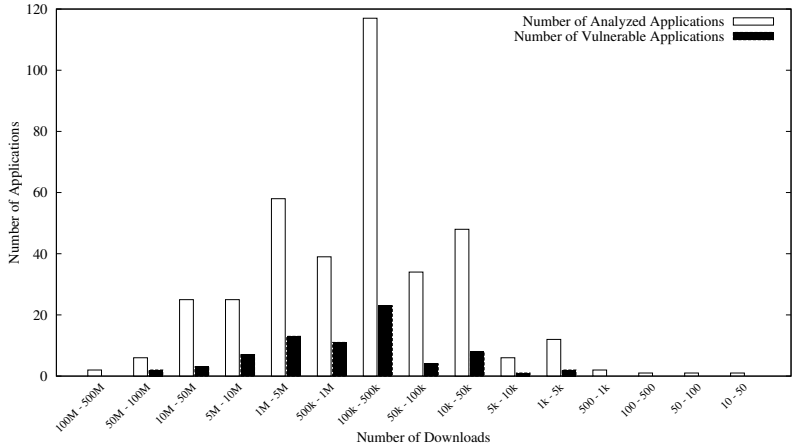


Fig. 4. Number of apps susceptible to data manipulation vs. number of analyzed apps, divided by popularity.

Looking closely at the vulnerable apps, Figure 5 shows the different file formats used to store the apps data on the internal storage of the Android OS. The Figure also shows that 52 vulnerable apps use XML, corresponding to about 70% of the apps, while the remaining 30% resort to using SQLite databases.

In the course of this study, we found that 87 of the vulnerable apps have logging options, and 23 of them, amounting to 26%, store user data in clear text in the application files. Consequently, applying the method described herein will reveal some sensitive user data, such as usernames and passwords. In some cases, we found that the app was downloading lists of usernames from other users of the app to the local storage via Internet.

Finally, the study also reveals that a greater percentage of the apps do not allow for application backup, implying that the `android:allowBackup` property on these apps is set to `false` in the `AndroidManifest.xml`, which thus protects them against the method described herein. Though disallowing the backup prevents the attack via the procedure described herein, it does not prevent manipulation when root access is available. Ideally, developers should correctly integrate appropriate security mechanisms, namely cryptographic related mechanisms.

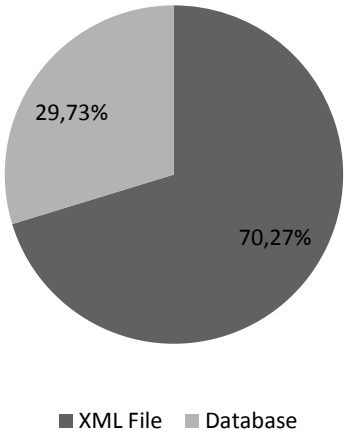


Fig. 5. Pie chart depicting the file formats used to store application data for the subset of susceptible apps.

5 Conclusions and Future Work

Insecure Data Storage, a type of security flaw explored in this study, has been highlighted by the Open Web Application Security Project (OWASP) as one of the top 10 vulnerabilities that can be exploited in order to break into mobile apps. The study specifically focused on application data manipulation that can enable users or malicious entities to access paid features of an application for free, or even obtain sensitive user data, such as username and passwords. The newer versions of Android OS allow users to back up apps on a computer, and this feature can be misused to manipulate the apps in order to gain access into unauthorized functionalities or user personal data. The method described in this paper does not require the system to be rooted, and even a user with little expertise in the area can use it. There are actually some exploits that automate the whole process of downloading a specific app to the computer, change it, and upload it back to the device based on the method used herein.

In order to determine the extent of the problem, 693 apps with in-app purchase possibilities were downloaded from Google play, and after applying some selection criteria, 377 from the initial data set were considered for the final analysis. When the method described herein was applied on these apps, a total of 74 apps were either found to be susceptible to data manipulation or sensitive user data on such apps was available in plaintext. The results of this study show that the vulnerability does not depend on the popularity of an application, since the vulnerability was found on both popular and non popular apps. We argue that disabling the backup feature for such apps does not constitute a perfect solution to the problem, and recommend that developers should follow security best practices, which include ensuring data integrity and adopting efficient encryption mechanisms.

Planned future work and improvements include subjecting other smartphone OSs that allow users to back up apps on a computer (e.g., iOS) to a similar assessment. Another possible way this work can be investigated further is to consider expanding the data set, by taking other types of apps into account. Finally, automating the method described in this paper is another line of work that is worth considering.

References

1. Christian Håland: An Application Security Assessment of Popular Free Android Applications. Master's thesis, Norwegian University of Science and Technology (2013)
2. Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M., Conti, M., Rajarajan, M.: Android security: A survey of issues, malware penetration, and defenses. Communications Surveys Tutorials, IEEE 17(2), 998–1022 (Secondquarter 2015)
3. Felix Matenaar, Patrick Schulz, Mark Schloesser, Andreas Galauner: dexter (2015), `http://dexter.dexlabs.org/`, accessed Jun. 2015
4. Garg, R., Telang, R.: Inferring app demand from publicly available data. MIS Q. 37(4), 1253–1264 (Dec 2013)
5. James King: Android Application Security with OWASP Mobile Top 10 2014. Master's thesis, Luleå University of Technology (2014)
6. Joe Security LLC: Automated Malware Analysis - APK Analyzer (2015), `www.apk-analyzer.net`, accessed Jun. 2015
7. Knorr, K., Aspinall, D.: Security testing for android mhealth apps. In: Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on. pp. 1–8 (April 2015)
8. Mulliner, C., Robertson, W., Kirda, E.: Virtualswindle: An automated attack against in-app billing on android. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security. pp. 459–470. ASIA CCS '14, ACM, New York, NY, USA (2014)
9. OWASP: Projects/OWASP Mobile Security Project - Top Ten Mobile Risks - OWASP (2014), `https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks`, accessed Nov. 2014
10. Pieterse, H., Olivier, M.: Android botnets on the rise: Trends and characteristics. In: Information Security for South Africa (ISSA), 2012. pp. 1–5 (Aug 2012)
11. Shuai, S., Guowei, D., Tao, G., Tianchang, Y., Chenjie, S.: Analysis on password protection in android applications. In: P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014 Ninth International Conference on. pp. 504–507 (Nov 2014)
12. Vienna University of Technology, Secure Systems Lab / Computer Security Group at UCSB: Anubis - Malware Analysis for Unknown Binaries (2015), `https://anubis.iseclab.org`, accessed Jun. 2015
13. Vigário, F., Neto, M., Fonseca, D., Freire, M., Inácio, P.: Assessment of the susceptibility to data manipulation of android games with in-app purchases. In: Federrath, H., Gollmann, D. (eds.) ICT Systems Security and Privacy Protection, IFIP Advances in Information and Communication Technology, vol. 455, pp. 528–541. Springer International Publishing (2015)
14. Wegilant: Appvigil - Cloud based Android App Security Scanner (2015), `https://appvigil.co`, accessed Jun. 2015

15. Wei, T.E., Mao, C.H., Jeng, A., Lee, H.M., Wang, H.T., Wu, D.J.: Android malware detection via a latent network behavior analysis. In: Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on. pp. 1251–1258 (June 2012)
16. Xiao, Z., Xiao, Y.: Security and Privacy in Cloud Computing. IEEE Commun. Surveys Tuts. 15(2), 843–859 (2013)

Open-Source SDN switching platform evaluation

Pedro Gonçalves[1],Elson Costa[1], José Bonifácio[1], Paulo Pedreiras[2]

{pasg, elsoncosta, jnbonifacio, pbrp}@ua.pt
[1]ESTGA/IT, Universidade de Aveiro
[2]DETI/IT, Universidade de Aveiro

Abstract. IIoT (Industrial Internet of Things) cloud applications require reliable, fault-tolerant networks, supporting real-time guarantees and eventually allowing interaction with other applications already existing in the datacenters. The Software Defined Networks (SDN) paradigm is especially suited for the management of the network cloud because of its fine-grained admission control and management flexibility provided by the centralized resource management.

This paper presents an experimental assessment of two existing open-source SDN switch platforms, in order to evaluate its suitability to support IIoT applications. Tests include assessment of fault-tolerant capabilities, computation requirements, session setup times and QoS capabilities of the SDN switches.

1 Introduction

Industrial Internet is a new area of research that ambitions to bring Internet technologies contributions for industrial processes, improving flexibility of manufacturing equipment, its efficiency and reducing the cost of installation and maintenance of communication platforms. It assumes that manufacturing process sensors and actuators are connected by Ethernet-based systems to the factory management systems, which, in turn, are connected with more powerful data processing systems, usually housed in Cloud platforms.

Those platforms usually apply virtualization technologies in order to adapt computing power demand, dynamically, allocating virtual machines and network resources over the data centers' physical resources.

Software Defined Networks (SDN) is a new network management paradigm, creating a centralized resource management point, able to dynamically apply more complex network resource management policies and mechanisms. SDN networking offers a programmatic network management process, defining policies that improve network efficiency and safety, offering enhanced Quality of Service (QoS) guarantees to services, or even creating fault-tolerant networks. The fine grain and centralized network management approach brought SDN technology into cloud networks, allowing the integration of industrial and generic information technologies in the same management platforms [1].

Even though much work has been developed in recent years around SDN [2][3] [4], both by academia and the industry, available solutions are usually focused on the development of controller algorithms, disregarding the lowest layer of management infrastructure. In this context SDN switching solutions usually depend on commercial implementations, usually extremely expensive, that despite implementing an open technology are presented as a black box, not allowing any sort of modification.

This paper presents an assessment of two existing open-source switching platforms, carried out in order to evaluate their adequacy to support the specific requirements of industrial applications. On this study, it is evaluated the performance of each one of the switches in terms of latency, computational requirements and the time required to setup new flows. Moreover, the paper also presents results concerning fault-tolerance and QoS mechanisms.

The rest of this paper is organized as follows: section 2 describes the Industrial Internet environment, while section 3 presents background on Software Defined Network technology. Section 4 presents the SDN switching platforms under evaluation and the test environment, while section 5 describes the tests carried out and presents their results. Finally, section 6 concludes the paper.

2 IoT networks on industrial environments

Over the last years, most countries focused their attention on the industry recovery and in improving their efficiency through the use of ICT technologies in industrial environments, launching novel initiatives and programs. The Industrial Internet was firstly introduced in a 2012 [5], envisaging the integration of industrial machines with corresponding data systems using Big Data analytics techniques, allowing remote and centralized visualization, as well as the establishment of physical-human networks in order to ease the cooperation between humans and robots. *Industrie 4.0* [6] and *La nouvelle France industrielle* are two European examples of industry dynamics, in line with IIoT paradigm. Moreover, China proposed its *Made in China 2025* strategy to promote domestic integration of digital technologies and industrialization. High-level dialogue between the German and Chinese governments, on how the two manufacturing powerhouses could work together to accelerate the realization of the Industrial Internet in their two countries, has also been taking place. Several consortia, notably as the *Industrial Internet Consortium (IIC)*, the *AllSeen Alliance and Open Interconnect Consortium (OIC)*, emerged to address the growing need for collaboration on common concerns such as security and interoperability.

Industrial networks are tightly tied to CPS systems, coupling computational and physical elements, often forming a feedback loop where physical processes affect the computations and vice-versa. This tight coupling presents strict requirements in terms of predictability, latency, dependability and security.

Although both require the same reliability from the network, computational processes existing in the cloud and control applications from industrial plants have different communication requirements. Industrial plant IP backbone applications require strict timeliness, in contrast with generic cloud network applications.

IIoT networks follow, in general, the IoT network architecture and network elements. Physical and industrial processes are controlled through a set of sensors, actuators and cyber-physical systems, which are interconnected to the IP world via gateways.

The IP backbone connects the field level gateways, with local management systems and with the Internet world by means of Internet gateways. These allow the communication with other industrial facilities, as well as with Cloud platforms where smarter, powerful algorithms, associated with Big Data processing, take place.

The connection between CPS and Cloud platforms allows to vertically interconnect and embed production systems with economic processes, and to combine CPS systems horizontally in real-time networks.

Similar to the broader IoT market, the Industrial IoT market requires inexpensive nodes and communication technologies that are easy to work, install and manage. Given the characteristics of industrial processes, and the fact that these processes typically involve mechanical, physical and chemical safety-critical activities, industrial networks pose other specific requirements, mainly related with performance and reliability.

3 Software Defined networks

SDN [1] is a network management paradigm that came from campus networks, and soon spread to other environments. It decouples the network control system (control plane) from the underlying network hardware responsible for forwarding data packets to their destination (data plane).

OpenFlow [7] is the most popular SDN technology which provides a standard API to communicate with network equipment and instructs it how to behave upon certain traffic classes/patterns, via the Controller entity. With SDN, network configuration is centralized and greatly simplified, and complex protocols traditionally used to shape the network operation and management can traverse the network with a lean programming environment, following a softwarization of networking procedures approach.

SDN is being adopted, amongst other reasons, as a mean to achieve hierarchical virtualization of network resources [8]. Nevertheless, SDN based solutions for Ethernet resource management [9][10] are usually sectorial, just addressing one management issue a time. For example, in [11] a proactive fault tolerant network management solution has been developed, making use of alternative routes and a mechanism to notify link breakdown events, delivering network packets even under random link failure conditions. Despite effectively solving the fault-tolerance issues, the work did not consider any aspects related to the parsimonious use of network resources nor with QoS mechanisms.

In [3] a reactive fault-tolerant resource management scheme for fat-tree [17] data center networks is presented, using flat IP addressing. The solution uses multipath load balancing techniques, but doesn't consider QoS requirements, absolutely necessary in industrial scenarios. Moreover, the associated results present a considerably high recovery time, which is not fast enough no maintain TCP sessions over link fail-

ures. Similarly, [4] presents a reactive fault-tolerant management solution that performs link failure detection and new topology establishment for fat-tree data center networks, but it does not consider multipath nor network resource optimization. Its results obtained in *Mininet* emulation tests, show even worse network restoration times, which are associated to Floodlight [9] controller.

A vast amount of work has been carried out on the Controller side [12][13], finding new schemes to organize and simplify the northbound API that abstracts SDN details to network applications.

Parallel to this, the coexistence of different applications, with different network requirements, such as the applications present in data centers, represents a huge traffic engineering challenge that requires a deep awareness of each application traffic profile. Unable to perform such a thorough traffic analysis, network administrators implement a protection measure consisting in overprovisioning network resources, maintaining a comfortable margin of free resources in the hope that they will be enough to transport information at unknown peak traffic conditions. However, this kind of practice does not favor anyone truly: the data center customer has no quality assurance for the contracted services, and the data center operator has a huge waste of network resources and energy consumption.

The inefficient use of network resources in the data center is reflected also in the selection of routes used by data streams, usually determined by the selection of the network paths between the source and the destination. Standard approaches usually culminate in the overloading of the shortest paths. As a result, resource management algorithms have to include load-balancing mechanisms.

4 SDN switch comparison

In order to choose a SDN switch, a survey of open-source SDN switch implementations was made, and the implementations were analyzed in terms of latency times, memory and processing requirements, as well as in terms of capability to implement QoS mechanisms.

4.1 SDN switches under evaluation

The most popular open-source virtual switches that are used in a SDN network, is the *Open vSwitch* and the *Ofsoftswitch13*.

Open vSwitch is a software implementation of a virtual multilayer network switch, which aims to provide a switching stack for hardware virtualization environments. It can be deployed natively over a Linux machine, or running within the virtual machine hypervisors, which made it to be ported to multiple virtualization platforms and switching chipsets. In order to increase stability and security, it presents gradual upgrades to new versions, which can take some time to have the ability to work efficiently with higher versions of the OpenFlow Protocol. However, all the stable releases have passed various quality tests making this switch a production quality virtual switch.

Ofsoftswitch13, was born in a research project carried on by a Brazilian partnership of technological companies and is now supported by Ericsson Innovation center/Brazil. It is a user-space software switch that was prepared to comply with Openflow version 1.3 features. It was designed to offer a simpler interface than *Open vSwitch*, facilitating the implementation of new functionalities by normal users.

4.2 Test setup and hardware

To enable a fair comparison of both SDN switches, a common platform was created for their implementations. In all the machines it is installed a Debian 7.0 distribution, with kernel 3.2.0-4-686-pae over an ext4 filesystem, and subsequently each one of the machines has been assigned a role in the network and received specific software. With the exception of interfering element, which is equipped with an onboard gigabit Ethernet interface, all the other machines have Fast Ethernet interfaces. TABLE 1 summarizes the key hardware characteristics.

TABLE 1 – Hardware summary

Element	CPU	Memory
SDN switch	AMD Athlon XP 2.6GHz	2,5GB
SDN controller	Intel Core 2 Duo 2.2GHz	4GB
Network probe	Core2Duo 3,16GHz	512MB
Interfering element	Intel Core I3 2.27GHz	4GB

5 Switching platforms evaluation

5.1 Faul-tolerance tests

The fail-tolerance mechanisms in a switch allow it to send the packets by an alternate link when the primary link fails, ensuring network connectivity. Both *open vSwitch* and *Ofsoftswitch13* claim to implement these mechanisms, so a very simple test network (Fig. 1) was created in order to test this mechanism.

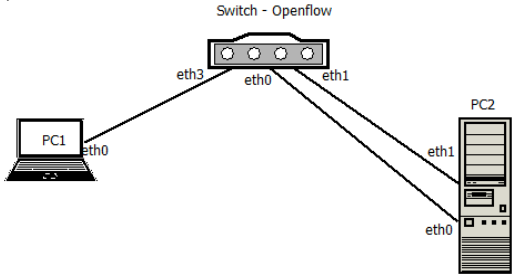


Fig. 1 – Fail-tolerance test network

Before simulating link failures, we created rules, with different priorities, to forward traffic to PC2 via different ports. The first rule, that had higher priority, defined packet forwarding by port eth0, while the second rule, with lower priority, specified forwarding the traffic to port eth1. Then, simulated communication link failures were injected both via console and by removing and reconnecting the Ethernet cables.

During these events the traffic was captured and the analyzed. The results show that in both implementations the switches were able to detect the failure on the main link and forwarded the traffic to PC2 via the lower priority link with minor delays. Moreover, no packet losses were detected. Reconnection of the link associated with the main route revealed an *ofsoftswitch13* bug, which ignores the main route link-up event and keeps forwarding traffic using backup route. Regarding *ofsoftswitch13*, the link-up and link-down event detection was made successfully and the main and back-up rules are used appropriately to forward traffic.

5.2 Latency

One of the most important features of a switch is the latency time, which has an additive effect on the overall network latency. In order to measure the switches' latency, a very simple network was mounted (**Fig. 2**), consisting of a two-port traffic generator and a switch that was configured to transfer the incoming traffic of one port to another port, to which the second interface of the traffic generator is connected. By receiving its own traffic by another port, the packet generator is able to measure the total latency without any other support mechanism, like synchronization.

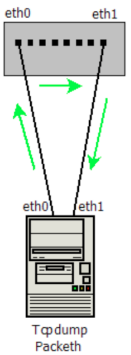


Fig. 2 - Latency measurement setup

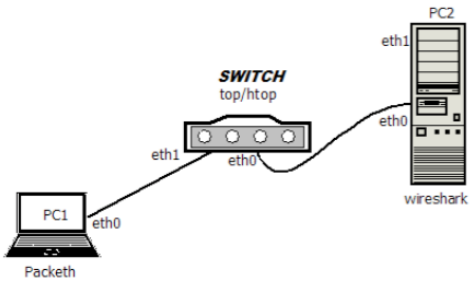


Fig. 3 – Memory and CPU requirements network

Tests consisted in sending 2000 bursts of 20 packets with a 10ms interval. The packet generator, the PC described in TABLE 1 enabled with Packeth, was configured to measure the time interval that the packets take to get back to the generator. In order to increase the measurement accuracy, PCs were initiated in run level 3, thus with the graphical subsystem disabled. Tests were repeated 10 times, and the meas-

ured values were processed to obtain the average, minimum, maximum and standard deviation. TABLE 2 summarizes the results.

TABLE 2. Latency measurements

Switch	Maximum time (us)	Minimum time (us)	Average time (us)	Stand. dev. (us)
Open vSwitch	1071	96	102	38.35
Ofsoftswitch13	1182	99	104	38.38

The figures in Table 2 show that *Open vSwitch* performs slightly better than *Ofsoftswitch13*, both in average and worst-case values. Moreover, it can be seen that the worst-case latency is relatively high, being slightly greater than 1ms, which can pose problems to some more demanding industrial scenarios.

5.3 Memory and CPU

The computational requirements of an application have a deep effect on the scalability of the solutions where it is used. In order to evaluate the computational requirements of SDN switches, the network illustrated in Fig. 3 was setup. Rules have been established in order to authorize flows from PC1 to PC2. During the transmission of these data flows the memory and CPU used by the packet switching process were measured. TABLE 3 summarizes the results of this test.

Tests revealed that *ofsoftswitch13* performed worse, both in terms of CPU and in terms of memory consumption. TABLE 3

TABLE 3 – Computational requirements

Switching platform	%CPU	%Memory	Process name
Open vSwitch	6,0	0,1	ovs-vswitchd
Ofsoftswitch13	8,0	1,2	udatapath/ofdatapath

5.4 Flow setup times

For the tests carried out to quantify the setup time for new flows the network topology illustrated in Fig. 3 was used, but now managed by the *OpenDaylight* controller. We started by disabling the *OpenDaylight* mechanism that automatically installs the rules according to the notifications sent by the switch about non-authorized flows. Then we started to send packet bursts during 30 seconds, spaced by a pause of two minutes. For succeeding packet bursts it were alternately defined and removed the rules that allowed traffic switching. Tests were repeated 20 times and traffic has been captured using *tcpdump* at the switch. Captures were analyzed, and TABLE 4 shows the average values.

TABLE 4 – Flow setup times

	Ofsoftswitch13		*Open vSwitch*	
Test	Flow-Mod	TCP (Ack)	Flow-mod	TCP (Ack)

Average (ms)	030.6282	668.861	9.450	40.080

5.5 QoS

In order to assess *Open vSwitch* QoS support, we setup the network illustrated in Fig. 4 and tested a set of OpenFlow policies related with QoS. Various approaches usual in network resource management were tried: setting minimum bandwidth, maximum bandwidth and defining different flow priorities.

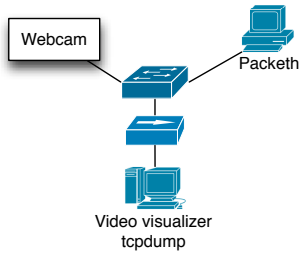


Fig. 4. QoS setup

Packeth was used in order to produce interference with the original data stream, competing with video packets on the access to switch queues. In order to reduce the number of machines in the setup, a direct policy definition approach was followed, avoiding the need of an OpenFlow coordinator. The rules were installed on switches to create and configure queues in the switch and define the flows that shall use them.

The first test carried out was meant to establish a baseline and consisted of establishing both sessions without any rule set, measuring how many packets from each session arrive to the destination. We observed that the video stream sent 1.7 Mbps and Packeth flow sent 7,9 Mbps. Additionally we could observe frequent video stops.

For the second test a rule that defines a minimum bandwidth value to both flows was set. However, since our goal was to protect the video stream, a very low minimum rate (1bps) to the interfering flow was defined. In the second test the video still had noticeable and frequent freezes and the bandwidths of 1,7 Mbps and 5,5 Mbps were measured respectively on the webcam and packet flows.

In order to ensure that the reserved bandwidth for the video stream was sufficient, the minimum rate assigned to it was increased to 5Mbps and we assigned to it an higher priority. As a result, we observed that video keeps low quality and captures demonstrated that the video stream sent 4,3Mbps and that Packeth sent 5,1 Mbps.

In the fourth test a new approach was tried, limiting the maximum bandwidth (3 Mbps) for the interfering flow. As a result we obtained a video without observable flaws and captures unveiled bandwidths of 5,02 Mbps for the WebCam flow and 2,2 Mbps for Packeth.

As a last test we defined limiting rules for both flows and a 4Mbps max rate was imposed to video stream. Captures exposed bandwidths of 3,9 Mbps from WebCam and 1,9 Mbps from Packeth flows. Moreover, in last test no disruption in the video was detected. TABLE 5 summarizes the results of various experiments.

TABLE 5 – QoS test results

Test	Switch policy	Independent link flows	Shared link flows	Video quality
1	No policy	WebCam - 5,5 Mbps Packeth - 9,8Mbps	WebCam - 1,7 Mbps Packeth - 7,9Mbps	
2	WebCam min-rate 4Mbps Packeth min-rate 1 bit		WebCam - 4,0 Mbps Packeth - 5,5 Mbps	
3	WebCam min-rate 5Mbps Packeth min-rate 1 bit Webcam with higher Priority		WebCam - 4,3 Mbps Packeth - 5,1 Mbps	Video frozen
4	WebCam min-rate 5Mbps Packeth min-rate 3Mbps		WebCam – 3,7 Mbps Packeth - 5,8 Mbps	
5	WebCam min-rate 5Mbps Packeth max-rate 3Mbps		WebCam – 5,02 Mbps Packeth – 2,2 Mbps	Video OK
6	WebCam max-rate 4Mbps Packeth max-rate 2Mbps		WebCam - 3,9 Mbps Packeth - 1,9 Mbps	No interference detected

The tests results enabled us to detect an improper implementation of QoS policies based on traffic differentiation, similarly to what is implemented in DiffServ. The provisioning of minimum rate rules was observed as helpfulness as well, since switches do not guarantee those rates as well.

Otherwise, we noted that the definition of maximum rate rules was a valid approach, allowing implementing on the switch a network admission controller, a scheme similar to a Policy Definition Point of COPS technology [14].

6 Conclusions

The SDN network management paradigm has been consecutively proposed for the management of data center networks, one of the components of IIoT networks, which according to IIoT architectures, should accommodate Big Data systems and backend control applications. Despite SDN network management to be centrally performed by controllers, switches perform a key role in SDN networks.

This paper documents a set of tests performed over two SDN switching platforms, as well their results. These tests included a fault-tolerance test, the measurement of switch latency, the analysis of the computational requirements and the measurement of the time needed to establish a session over switch queues. *Open vSwitch* performed better in all tests than *Ofsoftswitch13*, which probably is the reason why the software is so popular, and that made us decide in favor of this solution.

Once the switching platform was chosen, QoS tests were performed by checking if the switch was implementing the most often used techniques to ensure QoS. In this

analysis we detected that switches do not properly implemented traffic prioritization and that switches did not enforce the reservation of a minimum transfer rate for flows.

Obviously it is not possible to obtain with this switch the performance nor the reliability that one would get from a commercial switch. The selected platform should act as the basis for the development of a SDN prototype network, and will allow testing network management algorithms that could later be replicated in commercial hardware.

Acknowledgement

This work is funded by National Funds through FCT - Fundação para a Ciência e a Tecnologia under the project UID/EEA/50008/2013.

References

1. Leon-Garcia, A., Bannazadeh, H., Zhang, Q.: Openflow and SDN for Clouds. In: Cloud Services, Networking, and Management. pp. 129–152. John Wiley & Sons, Inc (2015).
2. Jo, E., Pan, D., Liu, J., Butler, L.: A Simulation and Emulation Study of SDN-based Multipath Routing for Fat-tree Data Center Networks. In: Proceedings of the 2014 Winter Simulation Conference. pp. 3072–3083. IEEE Press, Piscataway, NJ, USA (2014).
3. Ramos, R.M., Martinello, M., Rothenberg, C.E.: Data Center Fault-Tolerant Routing and Forwarding: An Approach Based on Encoded Paths, (2013).
4. Li, J., Hyun, J., Yoo, J.-H., Baik, S., Hong, J.W.-K.: Scalable failover method for Data Center Networks using OpenFlow, (2014).
5. Evans, P.C., Annunziata, M.: Industrial Internet: pushing the boundaries of minds and machines. (2012).
6. Drath, R., Horch, A.: Industrie 4.0: Hit or hype? [Industry Forum], (2014).
7. Pries, R., Jarschel, M., Goll, S.: On the usability of OpenFlow in data center environments. 2012 IEEE Int. Conf. Commun. 5533–5537 (2012).
8. Matias, J., Tornero, B., Mendiola, A., Jacob, E., Toledo, N.: Implementing Layer 2 Network Virtualization Using OpenFlow: Challenges and Solutions. 2012 Eur. Work. Softw. Defin. Netw. 2, 30–35 (2012).
9. Wallner, R., Cannistra, R.: An SDN Approach : Quality of Service using Big Switch ' s Floodlight Open-source Controller. Vol 35, 14–19 (2013).
10. Blefari-Melazzi, N., Detti, A., Morabito, G., Salsano, S., Veltri, L.: Information Centric Networking over SDN and OpenFlow: Architectural Aspects and Experiments on the OFELIA Testbed. Arxiv Prepr. (2013).

11. Goncalves, P., Martins, A., Corujo, D., Aguiar, R.: A fail-safe SDN bridging platform for cloud networks. In: Telecommunications Network Strategy and Planning Symposium (Networks), 2014 16th International. pp. 1–6. , Funchal, Portugal (2014).

12. Nunes, B.A.A., Mendonca, M., Nguyen, X.N., Obraczka, K., Turletti, T.: A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks, (2014).

13. Sezer, S., Scott-Hayward, S., Chouhan, P.K., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M., Rao, N.: Are we ready for SDN? Implementation challenges for software-defined networks, (2013).

14. Durham, D.: RFC 2748 (The COPS (Common Open Policy Service) Protocol). Req. comments. 1–38 (2000).

A Tool for Real-Time Assessment of IEEE 802.15.4 Networks Through Fault Injection *

Rui Pedro Caldeira, Jeferson L. R. Souza, Ricardo C. Pinto, and José Rufino

LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal
{rcaldeira, jsouza}@lasige.di.fc.ul.pt,
{ricardo.pinto, jmrufino}@ciencias.ulisboa.pt

Abstract. Advances in computer engineering and microelectronics have allowed the use of tiny and powerful computing platforms (i.e., sensors and actuators) everywhere, supporting the monitoring and control of, for example, process for industrial automation and functions within aerospace vehicles. Many of these systems have the ability to host, in the same computing platform, applications with different levels of criticality (or importance), i.e. mixed-critical systems. Wireless sensor and actuator networks (WSANs) become the vivid example of computer networks responsible for the monitoring and control activities of such systems. The dependability and the real-time properties of such networks are crucial. However, one key point is that WSANs are extremely susceptible to communication errors induced by electromagnetic interferences. Furthermore, there is a general lack of knowledge of such error patterns as well as no open tools enabling its capture. This paper presents a state of the art solution for one-hop assessment of WSANs in the presence of errors based on the IEEE 802.15.4 standard. The solution includes devices and functions to monitor the behaviour of the network as well as methods to emulate accidental errors and to perform intentional attacks. All these resources are managed and controlled by a customised version of the well-known open-source Wireshark network protocol analyser. This allows the generation of network error reports fundamental to the evaluation of the real-time capabilities of current wireless network protocols and standards. These error reports contribute to a better knowledge of the error characteristics of WSANs and therefore enable the design of more robust and resilient solutions for WSANs operation.

1 Introduction and Motivation

Wireless Sensor and Actuator Networks are one of the latest revolutions in networking. The absence of cables stem a reduction of Size, Weight and Power Consumption (SWaP) which combined with node mobility, lead to the adoption of these networks as a fundamental communication platform of many different types of systems that may host applications with different levels of criticality (or importance), which are usually

* This work was partially supported by FCT, through project PTDC/EEI-SCR/3200/2012 (READAPT) and through LaSIGE Strategic Project PEst-OE/EEI/UI0408/2014. This work integrates the activities of COST Action IC1402 - Runtime Verification beyond Monitoring (ARVI).

known as mixed-critical systems. Despite some advances in the support of real-time communication in wireless networks, there are still open problems that need the appropriate tools for their study and analysis. One key issue is that wireless networks are subjected to electromagnetic interferences from the surrounding environment that may impair ongoing communications. The presence of interferences can endanger the real-time guarantees of the communications as well as of the overall system. This paper discuses a highly flexible advanced open tool that allows the real-time assessment of one-hop IEEE 802.15.4 networks through the combination of network monitoring (e.g., for the capture of error patterns) with the emulation of accidental faults or even the injection of intentional attacks. The assessment of the network operation, and therefore the contributions of this paper, includes the facets that we detail next.

Network monitoring, which is a technique aimed at analysing the network interactions between its nodes and characteristics of operation such as reliability and timeliness, assumes a passive and non-intrusive role at network assessment. In particular, it allows to assess normal network operation and the disturbances in its behaviour in the presence of errors. Since particular error patterns may be specially relevant for the analysis of the network operation and their natural occurrence may be rare, there is the need to re-enact such error patterns, through fault injection.

Our fault injection methodology is aimed at directly inject interferences in the wireless transmission medium with the objective to stress test the network. Fault injection is split into two types: accidental faults and intentional attacks. The first type focuses in reproducing conditions that can be found in reality. This is very helpful in the sense that allows to test a network at will in a controlled environment, observe its behaviour, and stress the operation of network related protocols. The second is focused in controllably disturb a network, with a malicious objective. Objectives often revolve around data extraction, network unavailability or other actions relevant to the party conducting such disturbances. In this work we focus on accidental faults, and they can be achieved in two ways. Firstly, through the transmission of particular symbol patterns without obeying to the medium access control protocol in use. This translates to the injection of unrecognised noise. Secondly, in situations where it may be useful to also inject symbol patterns that are interpreted by the remaining nodes, we also support the injection of properly formatted traffic. Furthermore, combining network monitoring with fault injection allows to trigger a specific set of fault injection actions upon the occurrence of particular network traffic patterns. This coordinated action allows an active role in network assessment.

Network monitoring activities can be a very data-intensive task, specially when conducted concurrently with fault injection activities. Analysing the harvested data is simply too difficult to do without assistance. Wireshark [7], being a reference network protocol analyser, has the ability to create a graphical representation of network interactions, thus assisting in the network analysis task by being an integrated Graphical User Interface (GUI) for the whole suite.

The remainder of the paper is structured as follows. Section 2 reviews some related work. Section 3 describes the real-time assessment suite and its components. Section 4 describes the implementation details for the suite. Section 5 walks through some simple use cases and finally section 6 concludes the paper and describes the future work.

2 Related Work

Monitoring and fault injection are techniques often used in conjunction to perform dependability evaluation of computer systems and networks.

In a more theoretical approach, KleeNet [16] is a verification framework much similar to a simulation or model-checking. The framework acts as an Hardware Abstraction Layer (HAL) where the software to be deployed can run and be tested against a series of assertions that model the correct operation of the system. That said and reiterating the initial remark, all results remain theoretical and the behaviours of individual nodes as well as the medium are still based on models and to accurately model a complex environment such a Wireless Transmission Medium (WTM) is near to impossible.

Another interesting development is a hybrid architecture that allows to transfer state from test-bed nodes to simulation platform and vice-versa [14]. However, this approach is much complex because requires physical access to the test-bed hardware as well as access to a simulation platform that simulates both the computing hardware and the wireless transmission medium. Additionally to these requirements, middleware has to be developed in order to support the state transfer operations.

Due to WSANs growing popularity, researchers felt the need to create tools like the ones described earlier to assist them in various research activities. Regarding network monitors, popular use cases for this type of devices are, for instance, those described in [12,15]. However, these works focus in network performance evaluation, paying little to no attention to communication errors and to the stress of network operation via fault injection campaigns. In a theoretical fault injection approach and with network security in mind, strategies for the injection of attacks in wireless networks were devised. Xu et al. [20] raised concerns about how insecure WSANs are. A wide range of attack strategies were defined in [20] taking into account the scarcity of power resources in wireless devices.

O'Flynn [13] developed a dual-transceiver device capable of very precise attack injection techniques, but the control software has focused in approaches that maximize the reaction time in order to jam the highest number of packets possible and not in creating complex fault injection scenarios.

JamLab [6] is an addition to WSAN test-beds that allows to record and reproduce real interference patterns (or signals) as well as emulate some real world devices such as microwave ovens, IEEE 802.11 (Wi-Fi) and IEEE 802.15.1 (Bluetooth) devices. They also recognize that network robustness against interferences and packet loss is crucial to a correct functioning of the network and interferences can cause the timing constraints not to hold. Their study is focused in measuring the Received Signal Strength Indication (RSSI), the Link Quality Indication (LQI) and its impact in the Clear Channel Assessment (CCA). They proved that interferences have a great deal of impact in Packet Reception Rate (PRR) and if packet retransmission is required, power consumption is dramatically increased. JamLab, however, focuses in disturbing physical (PHY) layer operation by re-generating signals in the same frequency and with the same intensity as first detected. One major limitation of JamLab and of other studies that focus on physical layer interference [4,5,8], is that there are no simple methods of mapping such interferences into packet/frame losses. Our tool addresses this limitation by focusing on a packet/frame level approach.

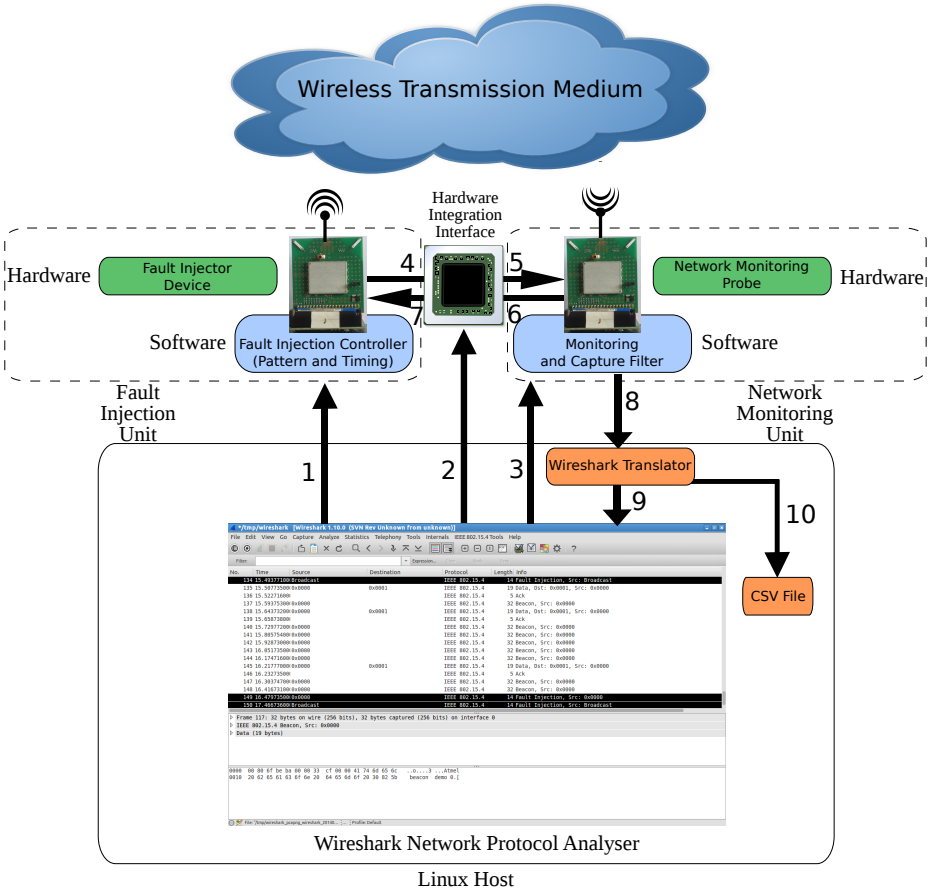


Fig. 1. Tool Architecture Design

3 A suite for Real-Time Assessment of IEEE 802.15.4 Networks

In this section we address in detail each component presented in our architecture and how they all integrate following closely the diagram in Figure 1.

This tool for the real-time assessment of IEEE 802.15.4 networks is composed by four components. Firstly, with the purpose of capturing and report network traffic, a network monitoring unit, which assumes the role of an error-aware packet sniffer was developed. All network monitors provide a very basic function of capturing network traffic without interacting with it. However, such capture is usually restricted to correctly formatted packets. Existing network monitors, to the best of our knowledge, do not capture and signal the occurrence of errors and this is one of the limitations we overcome with our design.

In contrast, fault injection devices can be described as devices that are able to interact with the underlying network but are not obliged to follow the medium access proto-

cols used in that particular network. In short, fault injection devices have the ability to interact with the network without any restrictions in regards of timing and/or content. And so, in order to provide the ability to emulate accidental faults and to perform intentional attacks, a fault injection unit has been built to flexibly support the injection of faults directly in the wireless transmission medium.

Thirdly, to provide interaction and cooperation between the network monitoring unit and the fault injection unit, as well to support advanced network analysis and filtering functions, a special purpose hardware integration interface was introduced.

Finally, Wireshark, a reference network protocol analyser with IEEE 802.15.4 packet visualization capabilities, was extended to manage all aspects of the network assessment, including control and data collection from the previously mentioned components.

3.1 Network Monitoring Unit

The Network Monitoring Unit includes a Commercial-Of-The-Shelf (COTS) hardware network interface device, acting as network monitoring probe (a.k.a. sniffer) with the purpose of capturing all traffic transmitted through a specific wireless radio channel. This is useful for testing networks and protocol implementations. The network monitoring probe is not an active member of the network, and is instead a passive listener. This is compliant with the IEEE 802.15.4 standard when using the so called *Promiscuous Mode* with the *integrity check disabled*: the network monitoring probe is allowed to capture and report all traffic following the IEEE 802.15.4 frame format specification. However, this is insufficient to assess the network error pattern since frames sensed with errors would not be specifically signalled and therefore would be handled as any other frame. Both correct and erroneous frames would be equally reported.

Wireless networks, due to the open nature of the transmission medium are extremely susceptible to electromagnetic interference (EMI) and therefore to frame errors. In order to capture and signal frames corrupted by errors, the operation of the network monitoring probe needs to be enhanced.

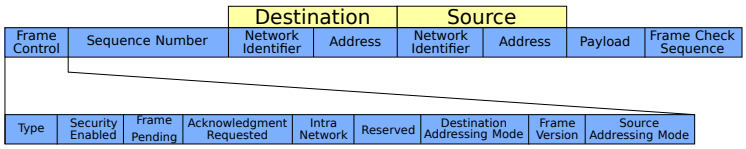


Fig. 2. IEEE 802.15.4 General Frame Format

The IEEE 802.15.4 traffic is constituted by frames following the general format represented in Figure 2. The Frame Check Sequence (FCS) field is a 16-bit frame integrity number used to evaluate the correctness of each captured frame. Deviations from the original content, e.g. resulting from corruption, can be detected through FCS checking mechanisms. The probability of undetected frame errors is negligible [9].

A simple, yet fundamental, *extension of this FCS integrity check mechanism* [18], enabled by modern network interface controllers such as the Atmel AT86RF232 [1],

allows to detect and signal erroneous frames. By taking advantage of the *FCS extension*, that allows us to signal corrupted traffic, and of the previously mentioned *promiscuous mode*, that allows us to capture all traffic, we are able to capture **all** network traffic that flows though the wireless transmission medium and to distinguish correct frames from those affected by errors.

This ability is **fundamental** to evaluate the error characteristics of a given network and can be used to estimate the impact of such error characteristics in the normal network operation.

To enable a precise evaluation of network operation timing characteristics, a timestamp is generated at the network monitoring unit at the arrival of each frame, including erroneous frames. This timestamp is attached to the frame and both are delivered at the Hardware Integration Interface, via the interaction 6 of Figure 1, and at the Wireshark translator, via the interaction 8 of Figure 1.

Concluding, the network monitoring unit offers to the surrounding components an extended promiscuous network traffic capture service that includes both correct and erroneous frames as well as the specific signalling of the latter.

3.2 Fault Injection Unit

The Fault Injection Unit is constituted by a hardware network interface, the fault injection device, controlled by a software component that runs on the fault injection controller, as illustrated in the left uppermost part of the Figure 1.

The fault injector device is a perfectly common Commercial-Of-The-Shelf (COTS) network interface with the exception that the network interface is configured to bypass the medium access control protocol thus allowing a direct access to the wireless transmission medium. Traditional IEEE 802.15.4 nodes use the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) mechanism to sense the medium before transmitting, which is therefore disabled.

Disabling the medium access protocol allows effective fault injection actions. This way, the fault injector device supports the injection of user-defined:

- Pure noise patterns;
- Selected data patterns, always preceded by the standard preamble[1];
- Selected data patterns encapsulated in a correctly formatted frame, thus injecting a standard compliant transmission.

The Fault Injection Controller is a software component executing on top of the hardware fault injection device with the responsibility to control the patterns and timings to be utilized in a fault injection campaign.

The fault injection unit has two modes of operation: user-defined configurations or pre-configured fault injection profiles.

Both modes originate the fault injection sequence illustrated in Figure 3 accordingly with the parameters that are now explained further.

[1] The preamble is a pre-defined sequence of symbols for synchronization of the receiver's circuitry with the incoming sequence of symbols.

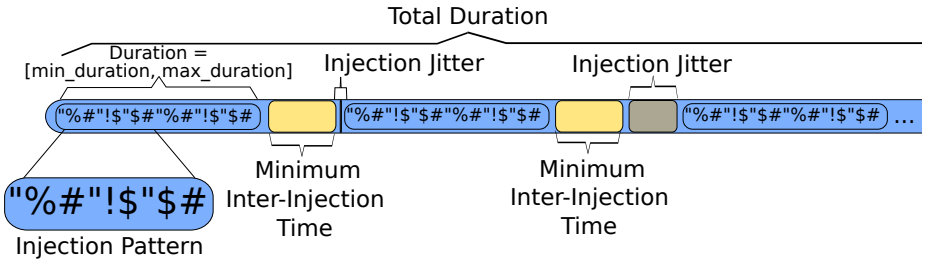


Fig. 3. Graphical representation of some fault injection parameters

Fault injection parameters User-defined configuration requires the specification of some parameters in order to build a particular fault injection profile. Some of the parameters are illustrated in Figure 3 and explained further. For future reference, and for the particular case of the IEEE 802.15.4, all time units are to be considered as being in microseconds.

- **Injection Mode -** The injection mode parameter determines how the potentially interfering data is sent to the wireless transmission medium. Firstly, when the noise mode is selected, the device simply sends a given sequence of symbols (injection pattern) to the wireless transmission medium until the injection event duration finishes. Secondly, in the preamble preceded mode, the user selected data is attached to the standard preamble and sent to the wireless transmission medium. Finally, in the encapsulated frame mode the user can configure all the fields of a correctly formatted frame;
- **Injection Pattern -** (see Figure 3) the format of the user-defined data, in hexadecimal, to be injected to the wireless transmission medium. Should a preamble be used (preamble preceded and encapsulated frame modes), it is attached before the injection pattern; otherwise (noise mode), only the injection pattern is sent. If the defined duration is lower than the time necessary to transmit the pattern this should be trimmed and periodically re-transmitted otherwise;
- **Minimum Duration -** the minimum duration of a single fault injection event. It can be expressed in time units or in bytes;
- **Maximum Duration -** the maximum duration of a single fault injection event. It can be expressed in time units or in bytes. If the minimum and maximum durations are equal, the duration of the fault injection event is equal to the previously stated values. Otherwise, it is equal to a value dictated by a given statistical distribution between the minimum and maximum durations;
- **Number of Events -** number of total fault injection events in the fault injection campaign, after which the campaign shall terminate;
- **Minimum Inter-Injection Time -** (see Figure 3) the minimum time interval between consecutive fault injection events;
- **Maximum Injection Jitter -** (see Figure 3) maximum value of a positive random time to be added to the Minimum Inter-Injection time, dictated by a given statistical distribution;

– **Total Duration -** (see Figure 3) the total duration of a fault injection campaign. If the number of events is specified, the user no longer is able to set its value, since it is confined to a lower as well as an upper bound dependent on the specific number of events, its (real) durations, its minimum inter-injection time and its (real) injection jitter durations. Conversely, if the total duration is specified, the fault injection campaign lasts until the specified duration is reached. The Total Duration is an alternative to Number of Events parameter described earlier, and for that reason, both parameters cannot be specified for the same fault injection campaign;
– **Trigger -** the condition to start the fault injection campaign using the previously described parameters. The trigger can be defined to operate according two modes: one-shot mode, where only one instantiation of the fault injection campaign is performed; and the cyclical mode, where the fault injection campaign is repeated cyclically until the stop command is explicitly issued.

Pre-configured fault injection profiles Some combinations of these parameters are so relevant that we have decided to include the capability to chose between a set of specific instantiations of the previously described parameters, thus defining pre-configured fault injection profiles. These profiles are useful both for the emulation of accidental faults and for the injection of intentional attacks.

– **Constant -** this profile continuously injects constant noise in the wireless transmission medium. This profile is meant for inducing communication blackouts such as jamming attacks [20];
– **Random -** this profile injects random noise on the wireless transmission medium. This profile is better suited for campaigns where it is only required the occasional corruption of transmissions [20];
– **Adaptive -** This profile tries to synchronize the fault injection timing with the traffic pattern of the underlying network. Thus the fault injection campaign is triggered by network monitoring unit after a specific type of traffic pattern has been detected by the monitoring and capture filter (see Figure 1);
– **Frame-type Adaptive -** specialisation of the **Adaptive** fault injection profile, where faults are injected to destroy the frames matching the specification(s) indicated to the network monitoring unit. This profile is specially useful when, for example, aiming to corrupt only beacon frames causing one-hop wide network blackouts [17].

A summary of the relevant fault injection parameters for each of the profiles is represented in Table 1.

One of the key features of this Fault Injection Unit is its flexibility when defining fault injection scenarios. The Fault Injection Unit is already equipped with several easy-to-use pre-defined scenarios while at the same time allowing a user to use configuration parameters if customisation is required. The presented fault injection profiles are suitable in the sense that they are practical implementations of well-known wireless network attacks which are also useful to emulate some accidental fault scenarios [20].

	Injection Mode	Minimum Duration	Maximum Duration	Total Duration	Number of Events	Minimum Inter-Injection Time	Injection Jitter	Trigger
Constant	Noise	∞	∞	∞	-	0	0	One-Shot
Random	Noise	Random		-	-	Random	0	One-Shot
Adaptive	Noise	19 bytes	133 bytes	-	1	-	0	Cyclical
Frame-Type Adaptive	Noise	19 bytes	133 bytes	-	1	-	0	Cyclical

Table 1. Fault Injection Profiles expressed as Fault Injection Parameters

3.3 Hardware Integration Interface

The hardware integration interface is a component that enables interaction and cooperation between the network monitoring unit and the fault injection unit, as shown in Figure 1.

The first aspect of this interaction concerns the analysis and filtering of captured frames. In essence any frame field, such as frame-type, addresses and some contents of the payload, can be subjected to analysis. Selected configurations of the Fault Injector Unit such as pre-configured adaptive and the frame-type adaptive fault injection profiles requires analysis of the captured traffic and in these occasions cooperation between both units is required to trigger a fault injection campaign, as shown by interaction 6 of Figure 1, where the network monitoring tool hands frames for analysis to the hardware integration interface, and interaction 7 of Figure 1 where the hardware integration interface decides to start a fault injection campaign.

The analysis of the captured traffic allows to detect relevant pre-configured traffic patterns to be used to trigger fault injection campaigns. An effective and highly flexible analysis and filtering of network traffic can be directly mapped into special-purpose hardware components, such as Content Addressable Memories (CAMs), integrated into Field Programmable Gate Arrays (FPGAs) [19]. The possibility of integrating these types of special-purpose hardware components with the tool greatly enhance the reaction capabilities of the tool in the sense that these components enables parallel comparison and matching of the captured traffic, against several specified frame patterns.

On the other hand, the start and the end of each fault injection event needs to be issued to the network monitoring unit to be timestamped, ordered, and inserted into the traffic flow to be delivered at the Wireshark Translator, represented by interaction 4 of Figure 1 where the notification is sent to the hardware integration interface and interaction 5 of Figure 1 where the notification is forwarded to the network monitoring unit.

3.4 Integration with Wireshark

Wireshark is a very flexible reference network analyser. It is extremely general in the sense that it can be extended by the means of plugins to analyse networks and protocols

that were not initially considered in its design and engineering. Wireshark can monitor traffic both from real and virtual devices such as networking hardware, regular files and even inter-process communication channels. Beyond that, the captured data can be saved for later analysis. Based on these reasons, Wireshark was chosen to be extended in order to incorporate the control functionalities of the network monitoring and fault injection units.

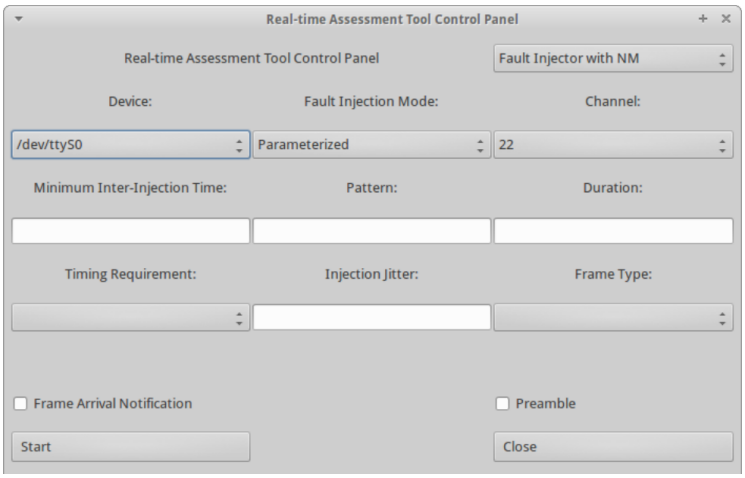


Fig. 4. Integrated Control Panel

Firstly, Wireshark communicates directly with the units in order to perform initialisation and command functions. The Fault Injection and the Network Monitoring Units are configured with the channel to be used in their respective activities, represented by the interactions 1 and 3 of Figure 1. In addition, the hardware integration interface is configured with all network monitoring and fault injection variables, represented by the interaction 2 of Figure 1, in order to correctly coordinate the interaction between the Network Monitoring and Fault Injection Units. In this sense, an extension to the Graphical User Interface was introduced to create a user-frendly interface to ensure that users can configure the units in an easy way. The result of this extension is presented in Figure 4. The figure presents the most complex case of the configuration of a fault injection campaign, accompanied with a network monitoring activity (Fault Injector with NM, in the top-right corner of Figure 4) with manual parameter specification (see section 3.2), including the path to the Network Monitoring Unit (below the *Device* label in Figure 4), and the path to the Fault Injection Unit which is derived as the device immediately following the Network Monitoring Unit. This establishes the communication paths between Linux/Wireshark Software Components and the underlying hardware units, as required for interactions 2, 3 and 8 (Network Monitoring Unit) and interaction 1 (Fault Injection Unit), depicted in the diagram of Figure 1.

After initialization, the network monitoring unit delivers captured frames to Wireshark indirectly. First, the captured frame arrives the Wireshark Translator (represented by interaction 8 in Figure 1) which is then adapted to the Wireshark format and promptly forwarded to Wireshark, as shown by interaction 9 of Figure 1.

The Wireshark Translator is a piece of software which has the responsibility to translate the raw frames collected from the network monitoring unit into data understandable by Wireshark. This process requires that a PCAP header [11], the format used by Wireshark, is attached to the raw frame so that Wireshark is aware of, among other aspects, the network type of the frame and its length.

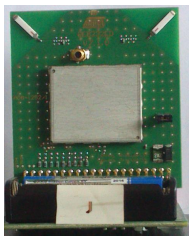


Fig. 5. Atmel REB232ED-EK

3.5 Statistics File

The Wireshark Translator application also creates a statistics file in the form of Comma-Separated Values (CSV), represented by interaction 10 of Figure 1, which is recognized by most statistical data applications. The file includes a timestamp, signalling when the frame starts and its duration, as well as the length of the frame, if the frame passed the checksum verification, and the frame itself. To simplify statistics file post-processing (e.g., inter-frame time and jitter calculation), a second timestamp signalling when the frame ends is also included. In this way, all relevant information regarding the frame is included in the same place. This file is useful in providing a base for additional processing that may not be achieved using Wireshark.

4 Implementation

Built using Commercial Off-The-Shelf (COTS) hardware, our implementation currently supports the Atmel REB232ED-EK (Figure 5) Evaluation Kit [3]. The hardware provides serial interface in order to connect to a computer host and exchange data.

Both units were implemented so they could be easily configured by Wireshark. In Figure 4, it is shown the integrated control panel. To configure a network monitoring session, it is required to select the path of the hardware, as well as the wireless radio channel to be monitored and the function in the top right corner of Figure 4 (e.g., standalone network monitoring or fault injection with network monitoring).

The monitoring starts by signalling the Network Monitoring Unit (via interaction 3 of Figure 1) and starting up the Wireshark translator. After a frame capture is successful, elements including the frame length and timestamps are attached to the frame and sent out via the serial connection (via interaction 8 of Figure 1).

In this work we choose to output the raw data and not commit to any specific data output format (e.g., the PCAP format) in order to save the scarce input/output resources of the Atmel hardware supporting both the Network Monitoring and the Fault Injection units and to allow any application to read the traffic and perform any kind of processing and analysis. The Wireshark translator has the function to conform the Network Monitoring Unit output to the PCAP format and forward it to Wireshark.

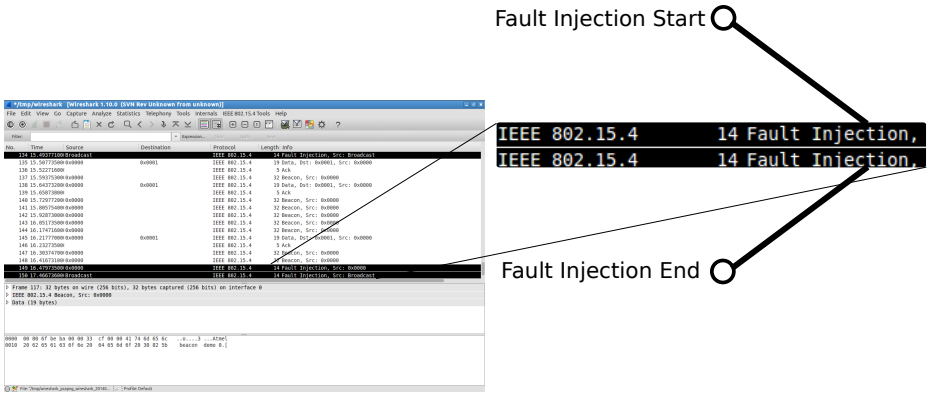


Fig. 6. Wireshark capture screen

With this control panel is also possible to define fault injection campaigns making use of the parameters or modes described in section 3.2. A fault injection campaign is constituted by a series of fault injection events. In the Wireshark screen (see Figure 6), additional elements (represented as black shaded highlighted frames) were introduced to Wireshark to delimit fault injection events. These elements, represented in Figure 6, notify the users when a fault injection event begins and when it ends, representing between them the frames that arrived during the fault injection. In the particular case of the example in Figure 6, the Constant fault injection profile (Section 3.2/Table 1) was selected. Since pure noise is constantly injected in the wireless transmission medium; no frame is received by any node, and therefore no frame is captured during the fault injection campaign. It is worth noting that the IEEE 802.15.4 interpretation capabilities of Wireshark were extended in order to make sure that these new elements were recognized and all functionalities of Wireshark, such as filtering and colouring, also apply to these new elements.

5 Use Cases

This section discusses two simple, yet illustrative use cases: firstly, an example emphasising the network monitoring functions while the second focuses in a fault injection campaign.

5.1 Standalone Network Monitoring

No.	Time	Source	Destination	Protocol	Length	Info
122	14.86852500	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000
123	14.99156800	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000, Bad FCS
124	15.11355700	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000, Bad FCS
125	15.23654600	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000
126	15.35955200	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000
127	15.48255400	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000
128	15.60554200	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000
129	15.72856400	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000
130	15.85153300	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000
131	15.97452500	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000[Malformed Packet]
132	16.09753900	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000
133	16.21952600	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000, Bad FCS
134	16.34256500	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000, Bad FCS
135	16.46555800	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000
136	16.58854900	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000
137	16.71155200	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000, Bad FCS
138	16.83456800	0x0000		IEEE 802.15.4	32	Beacon, Src: 0x0000
139	16.98456800	0x0000		IEEE 802.15.4	45	Beacon, Src: 0x0000, Bad FCS

Fig. 7. Standalone Network Monitoring

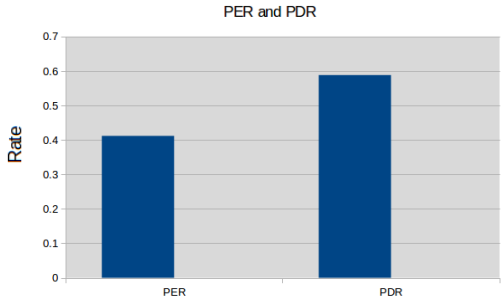


Fig. 8. Chart representing the Packet Error Rate (PER) and the Packet Delivery Rate (PDR) of a IEEE 802.15.4 network under heavy interference, as generated by the interpretation of the statistics file

The first use case, illustrates the functionality of our tool in the traffic monitoring of a IEEE 802.15.4 network. However, this network is operating in a ”dirty environment” where a heavy electromagnetic interference is expected from ”alien” wireless nodes.

One of the main sources of interference on the IEEE 802.15.4 channels from wireless ”alien” nodes results from the coexistence with IEEE 802.11 nodes. Since both standards operate in the same 2.4 Ghz Industrial, Scientific and Medical (ISM) bands the probability, in some cases, of both protocols interfere is high [10].

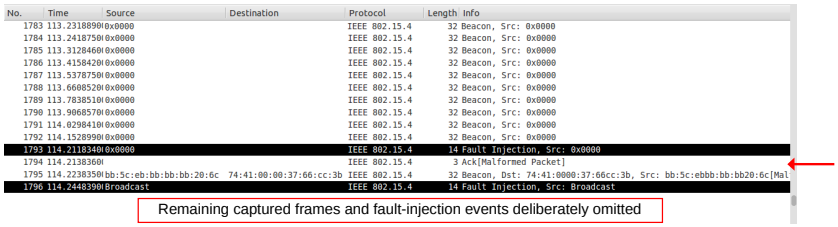


Fig. 9. Network monitoring with Fault Injection

To increase the likelihood of interference, we have set the IEEE 802.15.4 coordinator to channel 17 (2.435 Ghz) which is the closest channel to an ”alien” IEEE 802.11 access point operating in channel 6 (2.437 Ghz). The IEEE 802.15.4 network operates under very lightweighted load conditions, with the network coordinator transmitting mostly beacon frames.

The results from our analysis, inscribed in Figure 7, show a high number of corrupted frames, some of them grouped in bursts of interference. However, these error bursts do not violate (at least in this experiment) the upper bound of three consecutive frame errors defined in the IEEE 802.15.4 standard specification [2].

Furthermore, the statistics file generated was used to be interpreted in a LibreOffice Calc Spreadsheet equipped with functions to automatically parse the data and generate charts, like the one represented in Figure 8. This chart represents the Packet Error Rate (PER), calculated by dividing the number of corrupt frames by the total of captured frames, and the Packet Delivery Rate (PDR), calculated by dividing the number of correct frames by the total of captured frames, based on data shown in Figure 7.

5.2 Network monitoring with Fault Injection

In this experiment, intended to demonstrate the functionality and effectiveness of the fault injection methodology, the same IEEE 802.15.4 network operates in a ”clean environment” where only some occasional frame errors, due to the natural electromagnetic interference from the environment, are expected. However, a fault injection campaign was conducted, with the following parameters:

- **Injection Mode:** Preamble Preceded Symbols
- **Injection Pattern:** AABBCC
- **Minimum Duration:** 28000 microseconds
- **Maximum Duration:** 28000 microseconds
- **Number of Events:** 100
- **Minimum Inter-Injection Time:** 1000000 microseconds
- **Maximum Injection Jitter:** 0 microseconds
- **Total Duration:** Not Defined
- **Trigger:** Cyclic

After this parametrisation, the Fault Injection Unit transmitted the pattern *AABBCC* preceded by the IEEE 802.15.4 preamble in a loop during 28000 microseconds, then

waited 1000000 microseconds. This sequence happened 99 more times as stated in the Number of Events parameter. Simultaneously, the network monitoring unit listened to all these interactions then were passed on to Wireshark following the interactions 8 and 9 of Figure 1. An excerpt of the results of this fault injection campaign is illustrated in Figure 9, where the black backgrounded rows signal the begin and the end of a fault injection event. The beacon frame transmitted by the network coordinator and signalled by the red arrow at the right of Figure 9 has been corrupted by the fault injection event. Other captured frames and fault injection events have been deliberated suppressed from the screen of Figure 9, using the wireshark own filtering facilities. It is worth noting that in this context the frames designated as "Ack" in the Wireshark packet list window actually represents the fault injection. The AABBCC pattern when analysed by the Wireshark internal components (in particular, by the Wireshark dissector) is considered an Acknowledgement frame (see Figure 2).

6 Conclusion and Future Work

In this work we have presented a real-time assessment suite for IEEE 802.15.4 networks. This suite provides two services, namely network monitoring and fault injection.

Network monitoring allows the capture and analysis of network traffic under normal operating conditions and in the presence of errors. Each captured frame is timestamped with the arrival instant thus supporting the analysis of the real-time characteristics of the network and with an indication of correctness, thus supporting the analysis of the network error characteristics.

With this suite effective fault injection campaigns can be easily defined and instantiated while providing means to visualize and record its effects. Such fault injection campaigns are relevant, to test and validate models describing the operation of network related protocols and strategies to verify and/or improve it.

In particular, both services provided by the suite are useful to study and validate innovative research approaches aiming to bring hard real-time guarantees to WSANs. This has been one of the main motivations for designing and developing this tool.

Future work will include the sophistication of the hardware integration interface. The sophistication of the fault injection controller will allow more and better ways to specify a fault injection campaign. Finally, future work will also include improvements in the tools reading the statistics file, processing it, and outputting fault injection parameters that, when inserted in the fault injection tool, will emulate a scenario approximated to the one represented in the statistics file.

References

[1] *AT86RF232 - Low Power, Transceiver for Zigbee, IEEE 802.15.4, 6LoWPAN, RF4CE and ISM Applications*, 2011.

[2] IEEE standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (LR-WPANs). *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*, pages 1–314, Sept 2011.

[3] *REB232ED-EK - Low Power, Evaluation Kit for Zigbee, IEEE 802.15.4, 6LoWPAN, RF4CE and ISM Applications*, 2011.

[4] P. Bartolomeu and J. Fonseca. An assessment of the IEEE 802.15.4 PHY immunity to WiFi interference. In *Emerging Technologies and Factory Automation (ETFA), 2010*, Sept 2010.

[5] C.A. Boano, Zhitao He, Yafei Li, T. Voigt, M. Zuniga, and A. Willig. Controllable radio interference for experimental and testing purposes in wireless sensor networks. In *Local Computer Networks, 2009.*, Oct 2009.

[6] C.A. Boano, T. Voigt, C. Noda, K. Romer, and M. Zúñiga. JamLab: Augmenting sensornet testbeds with realistic and controlled interference generation. In *10th Int. Conference on Information Processing in Sensor Networks (IPSN)*, pages 175–186, April 2011.

[7] Gerald Combs. The Wireshark network protocol analyzer. Available at: `http://www.wireshark.org/`. Accessed in June 5, 2013.

[8] D. Eckhardt and P. Steenkiste. Measurement and analysis of the error characteristics of an in-building wireless network. In *Annual Conference of Special Interest Group on Data Communication (SIGCOMM)*, 1996.

[9] T. Fujiwara, T. Kasami, A. Kitai, and S. Lin. On the undetected error probability for shortened hamming codes. *IEEE Transactions on Communications*, 33(6), June 1985.

[10] I Howitt and J.A Gutierrez. IEEE 802.15.4 low rate - wireless personal area network coexistence issues. In *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, volume 3, pages 1481–1486 vol.3, March 2003.

[11] Van Jacobson and S McCanne. libpcap: Packet capture library. *Lawrence Berkeley Laboratory, Berkeley, CA*, 2009.

[12] A Koubaa, S. Chaudhry, O. Gaddour, R. Chaari, N. Al-Elaiwi, H. Al-Soli, and H. Boujelben. Z-monitor: Monitoring and analyzing IEEE 802.15.4-based wireless sensor networks. In *IEEE 36th Conf. on Local Computer Networks (LCN)*, pages 939–947, Oct 2011.

[13] C.P. O'Flynn. Message denial and alteration on IEEE 802.15.4 low-power radio networks. In *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, pages 1–5, Feb. 2011.

[14] F. Österlind, A. Dunkels, T. Voigt, N. Tsiftes, J. Eriksson, and N. Finne. Sensornet checkpointing: Enabling repeatability in testbeds and realism in simulations. In Utz Roedig and CormacJ. Sreenan, editors, *Wireless Sensor Networks*, volume 5432 of *Lecture Notes in Computer Science*, pages 343–357. Springer Berlin Heidelberg, 2009.

[15] W.-B. Pöttner and L. Wolf. IEEE 802.15. 4 packet analysis with wireshark and off-the-shelf hardware. In *Proceedings of the Seventh International Conference on Networked Sensing Systems (INSS2010). Kassel, Germany*. Citeseer, 2010.

[16] R. Sasnauskas, O. Landsiedel, M. H. Alizai, C. Weise, S. Kowalewski, and K. Wehrle. KleeNet: Discovering Insidious Interaction Bugs in Wireless Sensor Networks Before Deployment. In *Proceedings of the 9th ACM/IEEE Int. Conference on Information Processing in Sensor Networks*, IPSN '10, pages 186–196, New York, NY, USA, 2010. ACM.

[17] J. L. R. Souza and J. Rufino. Characterization of inaccessibility in wireless networks - a case study on IEEE 802.15.4 standard. In *Analysis, Architectures and Modelling of Embedded Systems. Proceedings of the Third IFIP TC 10 International Embedded Systems Symposium (IESS 2009), Langenargen, Germany, September, 2009.*, September 2009.

[18] J. L. R. Souza and J. Rufino. Analysing and reducing network inaccessibility in IEEE 802.15.4 wireless communications. In *38th IEEE Conference on Local Computer Networks (LCN)*, Sydney, Australia, October 2013.

[19] Spartan-3E FPGA family data sheet, August 2009.

[20] W. Xu, Ke Ma, W. Trappe, and Y. Zhang. Jamming sensor networks: attack and defense strategies. *Network, IEEE*, 20(3):41–47, 2006.

Enabling Wireless Sensor Networks with Contiki OS and CoAP

Ana Santos[1], Paulo Pedreiras[2], and Paulo Bartolomeu[3]

[1] DETI, University of Aveiro, Aveiro, Portugal
[2] DETI/IT, University of Aveiro, Aveiro, Portugal
[3] Globaltronic S.A., Águeda, Portugal

Abstract. The use of Web services in sensor networking applications will play a major role in the future M2M communications. The emergence of the Constrained Application Protocol (CoAP) has made possible the use of RESTful Web services in constrained nodes and networks such as, for example, Wireless Sensor Networks (WSNs). This paper presents an IP based solution to integrate sensors and actuators in smart lighting applications. The developed platform builds upon the widely researched Contiki embedded operating system. Besides detailing the system's architecture and implementation, this paper presents multiple results showing that the performance of CoAP based resource retrievals in constrained nodes is adequate for supporting networking services in street lighting networks.

Keywords: Internet of Things - Wireless Sensor Networks - CoAP - embedded Web services - Contiki

1 Introduction

Standing before the digital revolution of the 21st century, the Internet of Things (IoT) shall connect the digital world to the physical world. Trillions of embedded devices using the Internet Protocol (IP), also called smart objects, will be an integral part of the Internet. In this scope, wireless communication technologies, in particular WSNs, will play a major role. In order to support such an extremely large address space, a new Internet Protocol, called Internet Protocol Version 6 (IPv6) is being adopted. The IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN) has accelerated the integration of WSNs into the Internet. At the same time, CoAP has made it possible to provide resource constrained devices with RESTful web services functionalities.

This work builds upon previous experience in street lighting networks, for which a proprietary protocol, devised by the Lighting Living Lab [1], was implemented and used for several years. The proprietary protocol runs on a broad range of lighting control boards. For this work, the four channel LED driver (4LD) board was selected. In order to support heterogeneous applications with more demanding communication requirements and to improve the application development process, it was decided to port the Contiki OS to the 4LD board.

After porting the Contiki OS to the 4LD, the main goal is connect the 4LD nodes to a Wireless Mesh Network, so they can be reached from the Internet. The final goal is to actually provide Web services functionalities over IP, using CoAP. So this paper is aimed to prove that the performance of CoAP based resource retrievals in constrained nodes is adequate for supporting networking services in street lighting applications.

The remaining paper is organized in four sections: background, implementation, validation and conclusions. The background provides the basic concepts pertaining to Wireless Sensor Networks, the Contiki Operating System and the Constrained Application Protocol (CoAP). The implementation section describes the solution's hardware, the work required to adapt the Contiki operating system and the implentation of CoAP. The validation section examines several aspects of the implementation, namely the memory usage and network performance. An evaluation of the CoAP requests, in terms of response time, will also be performed. Finaly, the conclusion discusses the obtained results and presents several lines of future work.

2 Background

2.1 Wireless Sensor Networks

Wireless Sensor Networks (Fig. 1) consist of several small and highly power-efficient wireless devices capable of communicating sensor data using low-power and low-bandwidth links.

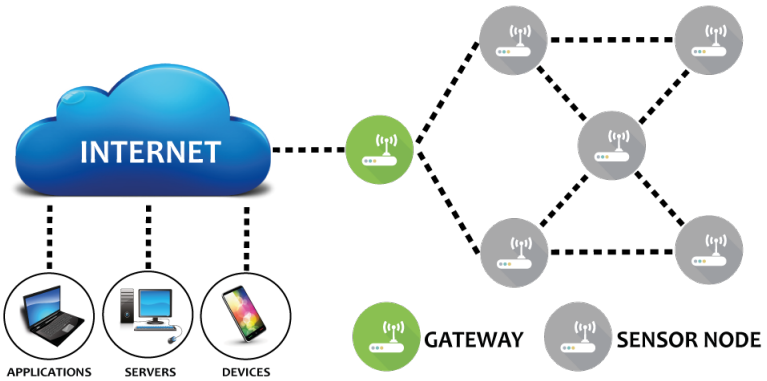


Fig. 1. Wireless Sensor Network.

Hardware Components Each sensor node of a WSN is defined both by its physical construction (the hardware) and by its behaviour (the software). The

typical sensor node encompasses the following hardware components [2]: communication device, microcontroller, set of sensors or actuators and power source.

Networking The majority of WSN solutions that currently exist do not provide support for IP. To provide interoperability between these WSNs and external networks is necessary to use specially designed gateways. Furthermore, the protocol used over the Internet these days, IPv4, can not be used due to being unable to address such a wide range of smart devices. With the emergence of IPv6, the address space available increased considerably in order to support billions of embedded devices. However, the complexity of providing IPv6 for devices with highly constrained memory and processing power has become a great challenge. The IETF has assigned two different working groups to integrate IPv6 in WSN devices: 6LoWPAN as the adaptation layer for IPv6 packets and RPL as a routing protocol for low-power and lossy networks (LLNs). The 6LoWPAN protocol provides a header compression and fragmentation mechanism for IPv6 that makes IPv6 packets more efficiently transmitted. The RPL specifies how to build a Destination Oriented Directed Acyclic Graph (DODAG) using an objective function and a set of metrics/constraints [3].

In the scope of this work, it is beneficial to use open-source software, since we want to be able to adapt the software to our specific needs without any additional costs. Currently, there are a variety of open-source implementations for WSNs. After an analysis of all existing implementations, we highlight the operating systems specially built for embedded systems, like the TinyOS and Contiki. These operating systems are open-source and provide implementations of the IEEE 802.15.4, IETF 6LoWPAN and RPL routing for several different memory and power constrained devices. Using an open-source operating system to implement our network will definitely offer more freedom in developing solutions for specific network requirements.

2.2 The Contiki Operating System

Contiki is a lightweight, portable and open-source OS written in C, specially tailored for WSN applications [4]. A typical Contiki configuration consumes 2 kilobytes of RAM and 40 kilobytes of ROM. Contiki is developed by a group from industry and academia lead by Adam Dunkels from the Swedish Institute of Computer Science.

The Contiki OS follows a modular architecture. At the kernel level it follows the event driven model, but it provides optional threading facilities to individual processes. The Contiki kernel comprises a lightweight event scheduler that dispatches events to running processes.

Communication Protocol Support Contiki was the first operating system that opened up the possibility of using wireless sensor nodes along with IP communications. This was achieved through the uIP, a TCP/IP protocol stack. Contiki's uIP stack is IPv6 Ready Phase 1 certified [5] and also provides 6LoWPAN support.

This stack was specially tailored to be used in resource constrained devices. Before uIP, the IP protocol was seen as too heavyweight to be used in these constrained devices. The existing implementations of the IP protocol for general computers would need hundreds of kilobytes of memory, however a typical constrained device offers only a few kilobytes of memory [2].

Contiki also provides an implementation of RPL (IPv6 routing protocol for low power and lossy networks) by the name ContikiRPL [6].

2.3 The Constrained Application Protocol (CoAP)

CoAP is a web application layer protocol that is specially designed for devices with constrained resources. CoAP is similar to the well-known and widespread HTTP, which is used throughout the Web. Both of these protocols use URIs to locate resources and share a common set of request methods: GET, POST, PUT and DELETE. This makes CoAP easy to understand and integrate into the current architecture of the Web [7].

CoAP messages CoAP uses the datagram-oriented UDP transport protocol to exchange messages. In terms of frame format CoAP has a 4 Byte fixed header and optional fields in Type-Length-Value (TLV) format as can be seen in Fig. 2.

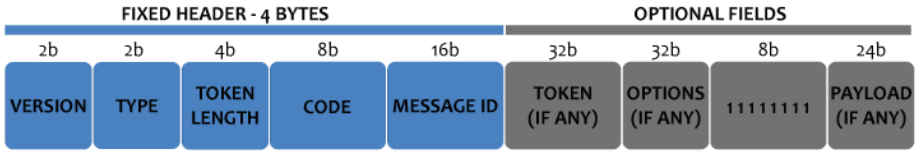


Fig. 2. CoAP frame format.

Different message types are used to establish a message exchange between client and server [7]:

1. **Confirmable (CON):** is used when a reliable transmission is needed. Since messages are transported using UDP, this reliability is achieved with packet retransmission if a response is not received within a given time out;
2. **Non-Confirmable (NON):** is used when a reliable transmission is not needed. It is quite useful for resources that are sent periodically;
3. **Acknowledge (ACK):** carries a response to acknowledge a CON request. This type of messages may carry response data or not;
4. **Reset (RST):** indicates that a CON message has arrived but there is no context to process it.

Blockwise Transfers When the resource representation is larger than what can be comfortably transferred in a single 6LoWPAN-based datagram, the CoAP protocol may use the blockwise transfers. Using this feature a single REST operation can be split into multiple message exchanges. This pattern results are very useful in terms of performance when a huge quantity of data needs to be transferred. Since the maximum payload size is very constrained in 6LoWPAN packets, applying even more network fragmentation could be even worse. So to avoid operations that could cause fragmentation at the network level, with the use of a blockwise transfers it is possible to carry the fragmentation of the data transmission from the network to the application layer [8].

Resource Observe Pattern In HTTP, the transactions are always initiated by the client. If a client wants to stay up-to-date about a specific resource status, it should perform GET operations again and again. In this type of networks, with limited resources, this pull model becomes very expensive. CoAP addresses this problem by providing a minimal enhancement to the REST model: adding the observer design pattern. Using this pattern, a client can indicate its interest in further updates from a resource by specifying the Observe option in a GET request. If the server accepts the option, the client becomes an observer of this resource and receives an asynchronous notification message each time it changes [9].

Resource Discovery In typical CoAP applications the devices must be able to discover each other and their available resources. This feature is already common on the Web, also called Web discovery in the HTTP community. Since were dealing with autonomous devices and embedded systems, the importance of uniform, interoperable resource discovery is much greater than on the current Web. To achieve resource discovery, CoAP servers are encouraged to provide a resource description available via the /.well-known/core URI. CoAP clients then perform a GET request on that URI, and the server will retrieve the description of all resources available [10] [11].

3 Implementation

3.1 Hardware

Two different hardware platforms were available for the experiments conducted for this work. One is the Giore platform, which was used to implement the gateway node. The other one is the 4LD platform, which is used as part of the wireless sensor network.

The Giore Platform The target hardware used to implement the gateway node is equipped with a PIC32MX795F512L microprocessor, a Microchip temperature sensor (TC1047AVNBTR) and an external antenna. The radio interface for the 868 MHz ISM frequency is the Hope Electronics RFM69HCW transceiver.

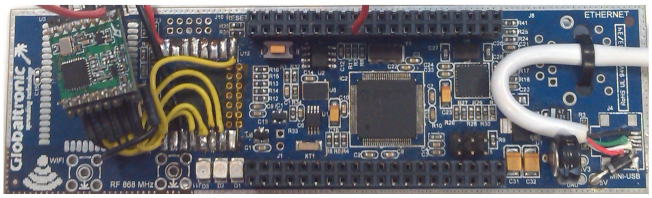


Fig. 3. The Giore board.

The PIC32MX microprocessor used has 524Kb of ROM, which includes 12Kb of Boot Flash, and 128Kb of RAM. The PIC32 is a 32 bit microcontroller with a maximum operating frequency of 80 MHz. The MCU does also include a UART interface, which can be used to communicate over a serial link. Besides UART, it also has a SPI interface, which is strictly necessary to communicate with the RF transceiver. A picture of the platform can be seen in Fig. 3.

The Giore platform is going to be connected through Serial Line Over IP (SLIP) to a computer running InstantContiki virtual machine. The computer has a Intel Core i3 processor and 4GB of RAM, which only 1GB is available for the virtual machine. This is more than enough for our purposes, and it should not force any restrictions on our solution.

The 4LD Platform The 4LD platform, as mentioned before, was developed by Globaltronic and is used to control street and industrial lighting systems. A picture of the platform can be seen in Fig. 4.

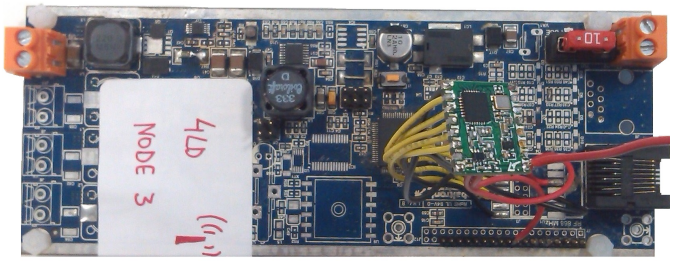


Fig. 4. The 4LD board.

The board is equipped with a PIC24FJ128GA308 microprocessor, a Microchip TC1047AVNBTR temperature sensor, a ST Microelectronics LIS2DHTR accelerometer and has also the Hope Electronics RFM69HCW RF transceiver. The main important feature is the 4 channels available to control LED lights. The PIC24FJ used is a 16 bit microcontroller with a maximum operating frequency of 32 MHz and has 128KB of ROM and only 8KB of RAM. The MCU, like the PIC32MX, also includes the UART and SPI interfaces.

3.2 Porting Contiki to the Hardware Platforms

Since a WSN might be composed of different hardware platforms which ideally have to run the same OS, it is clear to see why portability has become a crucial requirement for a WSN operating system. In this particular case we have two different platforms, already presented before, and the two have to run the Contiki OS. Thus, the first part of this work is port the Contiki OS to these platforms. Since Contiki doesn't offer support for the RFM69H transceiver used in this application, the second part is develop the RF transceiver device driver. In this section all the work needed to port the Giore and the 4LD platform is going to be described.

A General Port Before starting to present the specific details of the Giore and 4LD ports, first it is important to discuss the essential steps for porting a new platform to Contiki.

The first thing to do is to create a subdirectory in the /platform directory. In some ports it might be more useful to use as basis ports of platforms that are similar to the platform to be ported, whenever available. For example in the case of Giore the /platform/seedeye directory was used, since it contained the port to the SEED-EYE Board [12] (which is similar to Giore and it uses the same CPU - PIC32MX).

For every new type of CPU we need to create a subdirectory in /cpu directory, and some files, like *clock.c* for example, needs to be written. In our case, as already mentioned, the Giore platform uses the PIC32MX microprocessor and the Contiki OS already had this port implemented in /cpu directory. However, it was necessary to make some modifications to the initial port for the proper integration. In the case of the 4LD platform that uses the PIC24FJ microprocessor, it was necessary to write all the CPU port, because the Contiki didn't offer support for the PIC24. The PIC24 port will be described later in this section.

After this is done, the next thing to worry about is communication. We must ensure that the proper interface between uIP and low-level hardware code exists. In other words we need to write the device driver for the platform's transceiver. In our specific case, it was necessary to write all low level functions to interact with the RFM69H transceiver and adapt the device driver developed to both platforms used in this work.

In the end, in order to be able to run Contiki the *contiki-platformname-main.c* should be created. It just initializes the harware and then schedules and runs the processes defined. After this is done, we should be able to successfully upload the Contiki system on the new platform.

Porting the Giore Platform Porting to the Giore platform implied making some changes to the SEED-EYE port, however, no very low-level code changes had to be done. Basically the work done was towards system integration, in particular: fixing makefiles and performing modifications to various files in order to provide higher level functionalities. The resulting port is complete and incorporates all the functionalities of a Contiki system. The code for the port can be found in /platform/giore and in /cpu/pic32.

In the file *contiki-giore-main.c* is presented the *main()* function for the Giore board. This function calls the *pic32_init()* function to set the mode operation of the quartz oscillator which generates the clock signal used by the microcontroller timers with an oscillation frequency of 80 MHz. Then it also calls the *clock_init()* function defined in the file *clock.c*, which will set Timer 1 to generate an interrupt each 0.9766 ms. This period will be the lowest operating system unit time in the platform.

In order to unify the process of initialization of wireless communications was created on the platform Giore a function called *init_net()*. This function has defined the necessary steps to radio transceiver driver startup, as well as the definition of the platform MAC address, and IPv6 address assignment after TCP/IP library startup.

After this phase, the function *main()* also starts the process for the different OS Timers, as well as the processes listed in the *autostart_processes*. After executing this initialization phase the system enters in infinite loop and calls the function *process_run()* periodically. If the function *process_run()* returns zero, there are no pending events or processes in need of polling.

Porting the 4LD Platform Since at the time this project started in the community there were no ports for the 4LD microprocessor, the first phase of this port consisted in the porting of the Contiki OS to the Microchip PIC24 (PIC24FJ128GA308) MCU. The work developed here is also quite relevant to other research groups, since the Contiki will now support one more processor. The code for the 4LD port can be found in /platform/leddriver and in /cpu/pic24.

If we look into /cpu/pic24, we will find the PIC24 port developed. The module clock, available in this directory, is responsible for the measurement of system time. For this purpose it's defined a macro CLOCK_SECOND, which corresponds to one second in system ticks. A second on this platform corresponds to 64 system ticks, i.e. , a second elapsed corresponds to 64 interrupts generated by the microcontroller's Timer 1. In the case of the 4LD platform it is defined that the smallest unit of time has a resolution of approximately 15.6 ms .

If we look into /platform/leddriver, we will find the platform part of the port. In the file *contiki-leddriver-main.c* is represented the *main()* function for the 4LD platform. In this port, the function *main()* calls the *pic24_init()* function to set the mode operation of the quartz oscillator which generates the clock signal used by the microcontroller timers with an oscillation frequency of 32 MHz. This function also initializes the serial port, the process libraries and the process for the different OS Timers, as well as the processes listed in the *autostart_processes*.

3.3 The Network Stack

The network stack implemented in Contiki is a little bit different than the usual 5-layers model typically adopted in TCP/IP. Between the Physical and the Network layers we have 3 different layers: Framer, Radio Duty-Cycle (RDC) and Media Access Control (MAC).

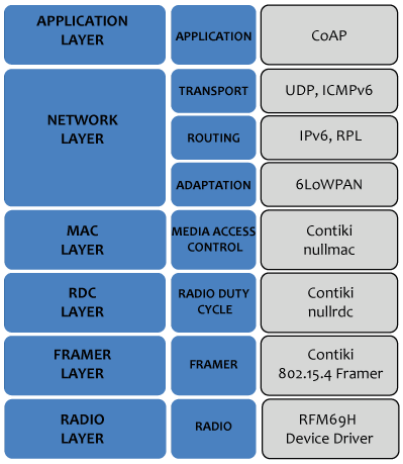


Fig. 5. The network stack implemented.

Physical, Framer, RDC and MAC Layers The physical layer corresponds to the RFM69H device driver developed.

The framer layer is responsible for creating frames with data to be transmitted and parsing received data. There are two types of framer layers: *framer-802154.c* and *framer-nullmac.c*. In our setup we used the framer-802154. This framer type creates and parses frames compatible with the standard IEEE 802.15.4.

The RDC layer is responsible to determine the sleep period of the nodes. Basically it decides when the packets must be transmitted and it also ensures that the node are awake when there are packets to be received from other neighbors. In our experiments we use the nullrdc module to ensure that the transceiver remains always on. The nullrdc does not save energy and only acts as a pass-through layer. Other RDC implementations in Contiki, such as ContikiMAC, provides an efficient management of the energy spent. Due to the initial phase of this implementation we opted for not using such modules, thus simplifying the performance evaluation of our network.

The MAC layer takes care of addressing and retransmission of lost packets. There are two types of MAC protocols implemented in Contiki: *csma.c* and *nullmac.c*. In our case we opted to use the nullmac module. This MAC layer is a simple pass-through protocol that simply calls the appropriate RDC function.

Network Layer In the network layer we used the 6LoWPAN protocol, so we can be able to use the IPv6 protocol implemented by the Contiki's uIPv6 stack. The network layer is configured with UDP and ICMPv6 protocols, and the routing protocol was the RPL [3].

The 6LoWPAN implementation in Contiki is an adaptation layer called SICSlowPAN [4]. Whenever a Contiki device receives an IPv6 packet, the MAC

layer will call the SICSlowPAN to adapt the packets to be used by the IPv6 layer (being implemented by the uIPv6 stack) and when uIPv6 needs to send an IPv6 packet also calls SICSlowPAN to adapt it for the IEEE 802.15.4 standard frames.

Since the Contiki implementation of 6LoWPAN does not provide mesh-under or route-over mechanisms [13], the routing is provided by the RPL implementation called ContikiRPL. ContikiRPL forwards the packets to the uIPv6 stack, providing routing tables based on the different objective functions selected [6].

Application Layer In our experimental setup, using as sensor nodes the 4LD platform and the Giore platform as border router, we used the Erbium (Er) REST Engine and CoAP implementation in the application layer. Erbium is a low-power REST engine developed for Contiki. It includes a embedded CoAP implementation and supports blockwise transfers and observing.

3.4 The Gateway

In order to be able to access the WSN over the Internet using the IPv6 protocol is necessary that one of the nodes in the network should be responsible for the task of interconnecting the WSN network and the IPv6 network. As can be seen in Fig. 6, the gateway implemented consists of two parts: the Border Router and the Base Station.

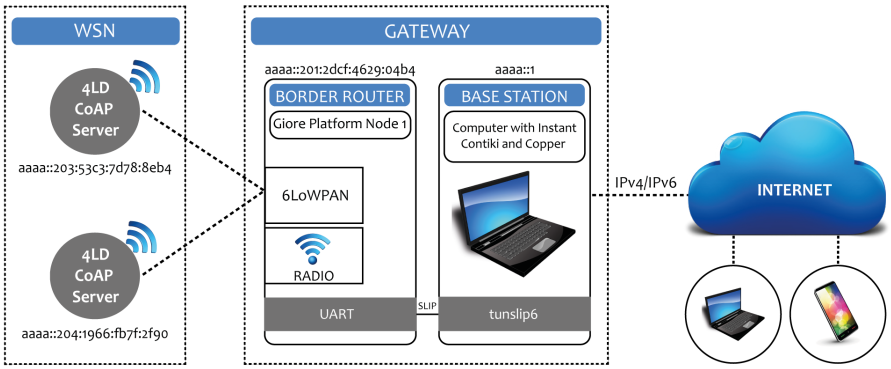


Fig. 6. Overview of the setup implemented using Erbium-CoAP and Copper Plugin.

The border router is responsible for converting between IPv6 and 6LoWPAN. IPv6 is used in almost every communication link, but the final step from the border router to the sensor node uses 6LoWPAN. The conversion is handled automatically by Contiki. The border router is connected with the base station

via a USB-dongle that provides a serial interface. Through a SLIP tunnel, the serial interface becomes the connection between the two parts of the gateway.

The SLIP tunnel is a part of Contiki. SLIP is used to encapsulate IP packets and send them over a serial link. Through the tool tunslip6 available in the Contiki, we can create a "tun" network interface on the base station with the address aaaa::1 on the LAN.

Using this two parts together, the IPv6 packets can be routed between the WSN network and the Internet. Thus the packets received over the SLIP tunnel will be forwarded and sent via wireless to the 4LD nodes. The opposite direction is also possible: data received on the wireless interface will be forwarded and sent over the SLIP tunnel to the base station.

3.5 The 4LD platform as CoAP server

As already mentioned before when presenting the network stack, we used in our implementation the Erbium REST Engine and CoAP Implementation in the application layer. In this experiment we used as basis the example code *er-example-server.c* available in the examples directory of Contiki OS. This code is a RESTful server example showing how to use the REST layer to develop server-side applications. In the client-side application we use the Copper Plugin for Firefox. Copper is an extension for Firefox to enable direct access to CoAP resources from a web browser. An overview of this setup can be seen in Fig. 6.

The REST engine defined in Contiki includes framework for developing both CoAP server and CoAP client applications. CoAP resources are easily defined using predefined macros. For each resource, we simply have to define its name, path, interface description, resource type and the actual code to provide the data. For periodic resources, we also have to provide the period and for actuator resources, we have to provide the callback function performing the action.

All the resources are statically defined and the respective functions are registered when the REST engine in the CoAP server starts. Each resource has to implement a handler function with the name [resource_name]_handler. For example, when the CoAP client sends a GET request to the CoAP server for the URI /temperature, the temperature_handler function will be called by the REST engine.

In the end, a typical RESTful Web service application consists of one main C-file that contains the resource macros together with their handler functions and one process that initializes the REST engine and activates the resources. In terms of resource discovery the CoRE Link Format is generated automatically for all resources. Our handler for the /.well-known/core URI respects chunk-wise processing and generates the required substrings without exceed the size of the buffer provided by the REST Engine. The application developed implements seven different resources, as can be seen in Fig. 7.

To interact with these resources we used the Copper (Cu) CoAP user-agent for Firefox [14]. This CoAP user-agent installs a handler for the CoAP URI scheme and allows users to interact with Internet of Things devices.

RESOURCE URI	FUNCTIONALITY
/.well-know/core	Resource Discovery
/device/info	Information about the 4LD node
/device/uptime	Information about the seconds elapsed
/actuators/led_toogle	Actuator that turns the red led on/off
/actuators/light_on	Actuator that turns the 4LD light on/off
/actuators/light_dim	Actuator resource that dims 4LD light
/sensors/temperature	Resource that returns the temperature

Fig. 7. Resources implemented.

To use Copper we only need to enter in the browser address bar: coap://[ipv6 address of the coap server]:udp_port. After opening the location of the CoAP server a click on 'Discover' button will retrieve the available resources from /.well-known/core. At a resource location we only need to use the buttons GET, POST, PUT, DELETE to perform the desired action. The response will then be displayed in the browser.

4 Evaluation of the implementation

4.1 Memory Usage

The memory usage is a crucial aspect of the implementation, thus should be analyzed. The results obtained can be useful to compare with other existing arcgitectures, but also to specify memory requirements when designing a new product. The Fig. 8 shows the Flash and RAM required by the firmware.

	FLASH (bytes)	FLASH (%)	DATA RAM (bytes)	DATA RAM (%)	STACK (bytes)
Giore Border Router	84456	16	10656	8	120360
4LD CoAP Server 1	69570	52	5706	69	2486
4LD CoAP Server 2	67839	51	5668	69	2524

Fig. 8. Firmware size comparison of the presented implementation.

4.2 Network: Performance Evaluation

The targets of the presented architecture are non-critical services. To evaluate if the system is capable to deliver a sufficient Quality of Service (QoS) we an-

alyzed the network response in terms of Round Trip Time (RTT) and Packet Loss (PLOSS) metrics. Using the experimental setup represented in Fig. 6, we evaluated the RTT and PLOSS between the Base Station and the 4LD Node making 1000 ping requests for each variation of the IPv6 packet payload. For this purpose we used the ping6 command that uses the Internet Control Message Protocol 6 (ICMPv6).

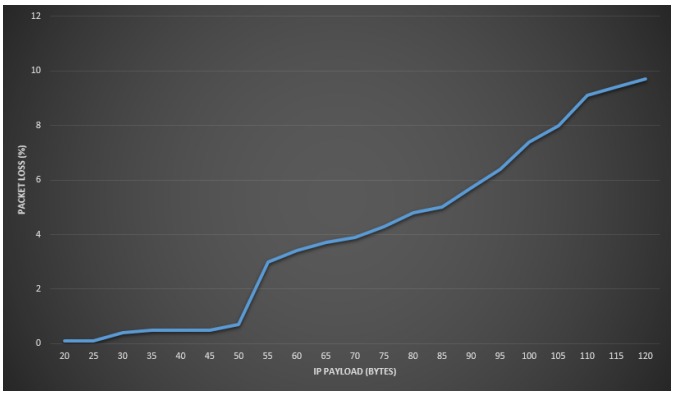


Fig. 9. PLOSS evolution according to ICMP payload size.

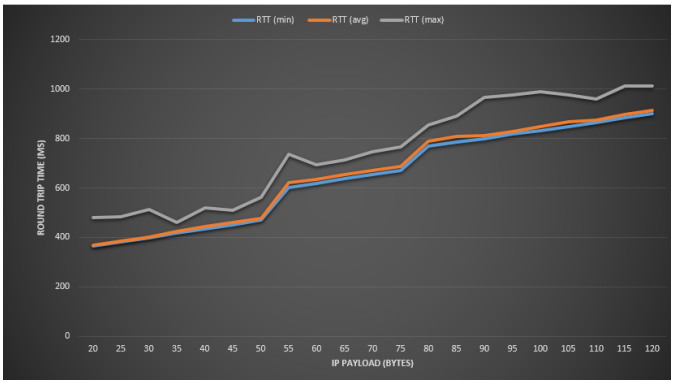


Fig. 10. RTT evolution according to ICMP payload size.

The Fig. 9 shows an increasing packet loss starting from 55 bytes of ICMP payload, which corresponds, including the 48 bytes of the IPv6 header, to a packet with a total of 103 bytes. The maximum packet size for IEEE 802.15.4 is 127 bytes and IPv6 requires that the underlying layers support packets of at least 1280 bytes. This means that the link layer fragmentation support must

exist. Taking into account that the IEEE 802.15.4 header has at least 25 bytes and the IEEE 802.15.4 maximum packet size is 127 bytes, we only can send a maximum of 102 bytes of data in one packet. So, if a packet payload has more than 102 bytes, the packet to be successfuly received must be fragmented. Hence we can conclude that the packet loss increase after the 55 bytes of ICMP payload is caused by the IP fragmentation mechanism.

Another characteristic that can be responsible for packet loss is related to the features of the considered devices having limited capacity for buffering packets at physical and MAC layers. In the case of the 4LD CoAP server the MAC layer can hold only 3 packets.

Despite this issue, solved by CoAP protocol through Blockwise Transfers pattern, the graphics clearly demonstrates that the 6LoWPAN networking stack implemented is working and that the experimental values of RTT and PLOSS are small enough to allow non-critical applications.

4.3 CoAP Transactions: Performance evaluation

This section presents some tests that have been made in order to evaluate the CoAP communication stack implemented. The Fig. 11 show a few results taken to evaluate the performance when retrieving different resources on the CoAP server. The retrieval time is measured on the base station to show the time taken to retrieve a given resource. From this figure it is clear that GET /.well-known/core is the resource request that takes more time so far. This was expected since it is the resource that has more bytes to be transmitted.

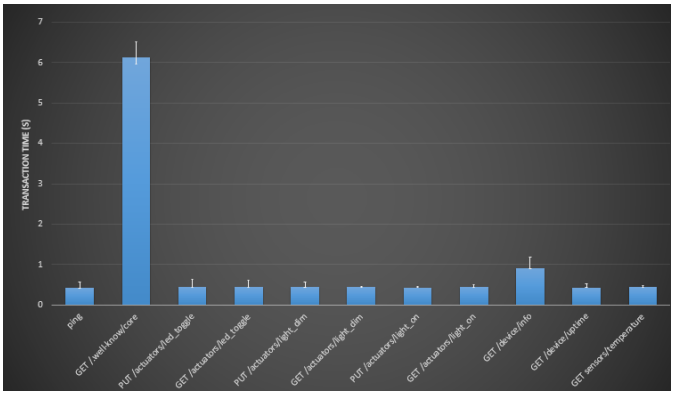


Fig. 11. Response time of the CoAP resource requests. The response time shown is the average result of 100 samples.

5 Conclusions

In this paper an architecture that supports WSNs applications in the field of Intelligent Street Lighting has been presented and evaluated. This architecture was built using a set of promising and recent protocols.

The preliminary results obtained are rather satisfying. The Contiki port to the PIC24FJ microcontroller along with a 6LoWPAN stack has been implemented and tested. The analysis shows that 6LoWPAN and RPL software available on Contiki, and also the remaining features available, matches the application requirements in a vast spectrum of application domains. Taking a look at the results obtained in Section 4, we can say that it is possible to deploy such a network in a large scale and develop complex applications.

The use of CoAP together with 6LoWPAN performs satisfactorily. It significantly reduces the size of the packets sent, which is important for wireless transmissions in resource constrained devices. An HTTP and TCP implementation would not have been possible to fit in our devices, specially in the 4LD board.

Choosing Contiki as the operating system has simplified the implementation a lot. It includes support for most of the protocols required on every layer. Implementing a network stack on our own would have been a very time-consuming task.

We can claim that, at least for non-critical applications, the presented architecture can be readily deployed and interconnected with a legacy infrastructure. Moreover, due to the scalability and versatility of the protocols used and the adoption of an IPv6 addressing for the sensor nodes, the presented architecture is easy to be maintained and upgraded in order to support new heterogeneous functionalities.

6 Future Work

Currently a new solution is being developed, using the Lightweight M2M protocol (LWM2M) in top of the existing CoAP implementation. The LWM2M defines a simple resource model where each piece of information made available by the client is a Resource. The Resources are further logically organized into Objects. The Objects/Resources are accessed with simple URIs: /Object ID/Object Instance/Resource ID.

The first release of the OMA LWM2M standard specifies an initial set of objects for device management purposes [15]. In the scope of this work, besides this set of objects two additional ones are going to be implemented: the IPSO Temperature Sensor and the IPSO Light Control defined by the IPSO Alliance.

This emerging protocol will enable large-scale M2M market growth by providing lower development and operational costs, greater flexibility, less fragmentation and faster time-to-market. Using a more open management standard, such as LWM2M, the market becomes more open with M2M devices being interoperable with different software management systems from different vendors.

References

1. Lighting Living Lab Home Page, http://www.lighting-living-lab.pt/, (2015)
2. Vasseur, Jean-Philippe and Dunkels, Adam: Interconnecting Smart Objects with IP: The Next Internet. Morgan Kaufmann Publishers Inc., San Francisco (2010)
3. Winter, T. and Thubert, P. and Brandt, A. and Hui, J. and Kelsey, R. and Levis, P. and Pister, K. and Struik, R. and Vasseur, J. and Alexander, R.: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks, RFC 6550 (Proposed Standard). (2012)
4. Contiki 2.6 Doxigen Documentation, http://contiki.sourceforge.net/docs/2.6, (2015)
5. IPv6 Ready Logo Program Approved List, https://www.ipv6ready.org/db/index.php /public, (2015)
6. Tsiftes, Nicolas and Eriksson, Joakim and Ko, Jeonggil and Dawson-haggerty, Stephen and Dunkels, Adam and Culler, David: ContikiRPL and TinyRPL: Happy Together. In: Proceedings of the workshop on Extending the Internet to Low power and Lossy Networks, (2011)
7. Constrained Application Protocol (CoAP), IETF Internet-Draft draft-ietf-core-coap, Rev. 18, Jun 2013, https://tools.ietf.org/html/draft-ietf-core-coap-18
8. Shelby, E. Z., Block-wise transfers in CoAP, Technical Report, Mar 2015
9. Hartke, Klaus: Observing resources in CoAP. Technical Report draft-ietf-core-observe-08.txt, IETF Secretariat, Fremont, CA, USA (2013)
10. Nottingham, M. and Hammer-Lahav, E.: Defining Well-Known Uniform Resource Identifiers (URIs), RFC 5785 (Proposed Standard). (2010)
11. Shelby, Z.: Constrained RESTful Environments (CoRE) Link Format, RFC 6690 (Proposed Standard), (2012)
12. Networks of Embedded Systems Group Home Page, http://rtn.sssup.it/index.php/software/contiki, (2015)
13. Chen Yibo and Kun-mean Hou and Haiying Zhou and Hong-Ling Shi and Xing Liu and Xunxing Diao and Hao Ding and Jian-Jin Li and de Vaulx, C.: 6LoWPAN Stacks: A Survey. In: Wireless Communications, Networking and Mobile Computing (WiCOM), 7th International Conference, pp 1–4, (2011)
14. Copper (Cu) CoAP user-agent for Firefox Home Page, http://people.inf.ethz.ch/mkovatsc/copper.php, (2015)
15. OMNA Lightweight M2M Object & Resource Registry, http://technical.openmobilealliance.org/Technical/technical-information/omna/lightweight-m2m-lwm2m-object-registry, (2015)

Índice de Autores