FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Departamento de Engenharia Electrotécnica e de Computadores

Qualidade de Serviço em Redes de Comutação de Pacotes

Rui Pedro de Magalhães Claro Prior

Licenciado em Engenharia Electrotécnica e de Computadores pela

Faculdade de Engenharia da Universidade do Porto

Dissertação submetida para satisfação parcial dos requisitos do grau de mestre em

Engenharia Electrotécnica e de Computadores

(Área de especialização de Telecomunicações)

Dissertação realizada sob a supervisão de

Professor Doutor José António Ruela Simões Fernandes,

Professor Associado do Departamento de Engenharia electrotécnica e de Computadores

da Faculdade de Engenharia da Universidade do Porto

Porto, Março de 2001

Resumo

Nos últimos anos tem-se verificado uma crescente tendência para a integração de novos serviços nas redes de comutação de pacotes, tradicionalmente usadas apenas para o transporte de dados. Entre estes serviços contam-se aqueles que tradicionalmente são disponibilizados em redes próprias, como o serviço de transporte de voz (telefone), e outros que apenas agora estão a emergir ou que dispunham de um suporte bastante limitado, como o transporte de vídeo. Estes novos serviços apresentam, no entanto, requisitos de tempo real que o serviço de dados não tem, e que as redes de comutação de pacotes como a Internet não foram concebidas para suportar. Para permitir essa integração é, pois, necessário introduzir mecanismos adicionais que permitam suportar os novos serviços com qualidade.

Esta dissertação aborda o problema da qualidade de serviço em redes de comutação de pacotes sob várias perspectivas. Iniciando-se com uma discussão do que é a qualidade de serviço, como pode ser caracterizada e quais as aproximações usadas para a implementar, continua com a apresentação de algumas tecnologias existentes para a suportar. Ainda sob um ponto de vista teórico são apresentados algoritmos de controlo de tráfego usados na diferenciação de serviços.

Sob um ponto de vista mais prático, é desenvolvido um ambiente de serviços diferenciados em TCP/IP inteiramente com base em *hardware* comum (computadores pessoais) e *software* de código aberto. É também desenvolvido um sistema de teste para este ambiente com base no mesmo tipo de ferramentas. Dada a anterior inexistência de documentação do suporte de controlo de tráfego do sistema operativo Linux, nomeadamente sob o ponto de vista da sua configuração com a ferramenta `tc`, é também incluida a sua documentação detalhada e completa, obtida a partir da análise do código-fonte.

Abstract

Recently we've been seeing a growing trend towards the integration of new services in packet switching networks, traditionally used solely for the purpose of data transport. Amongst these services are those traditionally provided over dedicated networks, such as the voice transport service (telephony), and others emerging right now or that previously had very limited support, such as video transport. These new services, though, have real-time constraints unbeknownst to the data transport service, which packet switching networks were not designed to support. In order for this integration to take place it's necessary to implement some additional mechanisms allowing for the new services to be provided with quality.

This thesis deals with the subject of quality of service from several points of view. It begins with the discussion of the quality of service concept, how it can be characterized, and the different approaches to it's implementation, and goes on with a description of some existing technologies for deploying QoS. Still from a theoretical point of view, some algorithms employed in service differentiation are presented.

From a more practical point of view, a testbed for differentiated services on TCP/IP is developed, based solely in common hardware (personal computers) and open-source software. Additionaly, a test system for this implementation is developed based on the same kind of tools. Given the previous inexistence of documentation for the traffic control support provided by the Linux operating system, namely from the point of view of configuration using the `tc` *utility, this thesis also includes a detailed and complete documentation, obtained from the analysis of the source code.*

Résumé

Depuis quelques années il se verifie une tendence croissante vers l'intégration de noveaux services dans les réseaux à commutation par paquets, traditionnellement utilisées seulement pour le transport de données. Parmi ces services il y a ceux qui traditionnellement sont fournis sur des réseaux à commutation de circuits, tels que le service de transport de voix (téléphonie), et d'autres émergeants ou qui avaient le support très limité, tels que le transport de vidéo. Ces noveaux services introduisent des contraintes de temps réel inexistantes dans le transport de données; par cette raison les réseaux à commutation par paquets n'étaient pas conçus pour supporter ces contraintes. Afin de faire cette intégration avoir lieu, il faut mettre en application quelques mécanismes supplémentaires pour que les noveaux services soient équipés de qualité.

Cette thèse traite le sujet de la qualité de service en réseaux à paquets de plusieurs points de vue. Elle commence par une discussion du concept de qualité de service, comment elle peut être caractérisée, les différents approches à sa mise en place, et une description de quelques technologies existantes pour la déployer. Encore du point de vue théorique y sont présentés quelques algorithmes utilisés dans la différentiation de services.

D'un point de vue plus pratique, un banc d'essai pour des services différenciés sur TCP/IP a été développé, basé seulement sur matériel commun (ordinateurs personnels) et logiciel à code source ouvert. Additionnellement, un systéme d'essai a été dévelopé basé sur le même genre d'outils. Parce qu'il n'y avait pas aucune documentation pour le support du contrôle de trafic fourni par le systéme d'exploitation Linux, cette thèse inclut également une documentation détaillée et complète de sa configuration en utilisant l'outil `tc` *, obtenue à partir de l'analyse du code source.*

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e símbolos

ABR	Available Bit Rate
AH	Authentication Header
API	Application Program Interface
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
BT	Burst Tolerance
CAC	Connection Admission Control
CAR	Committed Access Rate
CBR	Constant Bit Rate
CBQ	Class-Based Queuing
CDV	Cell Delay Variation
CDVT	Cell Delay Variation Tolerance
CER	Cell Error Ratio
CLIP	Classical IP (over ATM)
CLP	Cell Loss Priority
CLR	Cell Loss Ratio
CMR	Cell Misinsertion Rate
CODEC	Coder/Decoder *ou* Compressor/Decompressor
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check

CSZ	Clark, Shenker, Zhang — Proponentes do algoritmo
CTD	Cell Transfer Delay
DEC LAT	DEC Local Area Transport
DNS	Domain Name System
DWRR	Deficit Weighted Round Robin
ECN	Explicit Congestion Notification
ESP	Encapsulating Security Payload
EWMA	Exponentially Weighted Moving Average
FIFO	First In, First Out
FQ	Fair Queuing
GCRA	Generic Cell Rate Algorithm
GFC	Generic Flow Control
GPI	Generalized Port Identification
GRED	Generalised Random Early Detection
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISDN	Integrated Services Digital Network (v. RDIS)
ISP	Internet Service Provider
ITU-T	International Telecommunication Union - Telecommunication Standardization Sector
LAN	Local Area Network (Rede Local)
LLC	Logical Link Control
MCR	Minimum Cell Rate
MTU	Maximum Transfer Unit (Unidade Máxima de Tranferência)
NAT	Network Address Translation
NNI	Network to Network Interface

NPC	Network Parameter Control
NRM	Network Resource Management
nrt-VBR	Non-Real-Time Variable Bit Rate
OAM	Operations, Administration and Maintenance
OSI	Open Systems Interconnection
PCR	Peak Cell Rate (Débito de Pico de Células)
PTI	Payload Type Identifier
QoS	Quality of Service (Qualidade de Serviço)
RDIS	Rede Digital com Integração de Serviços (v. ISDN)
RED	Random Early Detection
RIO	RED with In/Out bit
RM	Resource Management
RSVP	Resource reSerVation Protocol
RTT	Round Trip Time (Tempo de Ida e Volta)
rt-VBR	Real-Time Variable Bit Rate
SCR	Sustainable Cell Rate (Débito Sustentável de Células)
SECBR	Severely Errored Cell Block Ratio
SFQ	Stochastic Fair Queuing
SMP	Symmetric Multi-Processor
SNA	Systems Network Architecture
SNAP	SubNetwork Access Protocol
SPI	Security Parameters Index
TCP	Transmission Control Protocol
TOS	Type Of Service
UBR	Unspecified Bit Rate
UDP	User Datagram Protocol

UNI	User to Network Interface
UPC	Usage Parameter Control
VC	Virtual Circuit (Circuito Virtual) *ou* Virtual Channel (Canal Virtual)
VCI	Virtual Channel Identifier (Identificador de Canal Virtual)
VLAN	Virtual LAN (LAN Virtual)
VP	Virtual Path (Caminho Virtual)
VPI	Virtual Path Identifier (Identificador de Caminho Virtual)
WF^2Q	Worst case Fair Weighted Fair Queuing
WFQ	Weighted Fair Queuing
WRED	Weighted Random Early Detection
WRR	Weighted Round Robin

Capítulo 1

Introdução

Recentemente tem-se verificado uma crescente tendência para a integração de diferentes serviços numa mesma rede. De facto, as vantagens de o fazer são óbvias, pois se existir uma infraestrutura única poupam-se os custos associados à instalação e manutenção de redes paralelas, cada uma dedicada ao suporte de um único serviço. Por outro lado, a explosão verificada em anos recentes na Internet, associada a uma crescente capacidade das ligações de dados, leva a que cada vez mais os utilizadores procurem conteúdos e serviços interactivos, nomeadamente de voz, áudio e vídeo, disponibilizados na rede global. Existe, no entanto, um entrave à integração generalizada de serviços com requisitos de tempo-real numa rede originalmente concebida apenas para o transporte de dados: a qualidade final do serviço é, normalmente, muito insatisfatória.

A pilha protocolar usada na Internet, TCP/IP, foi originalmente desenvolvida pelo *Department of Defence* (DoD) dos EUA para utilização na rede ARPANET (*Advanced Research Projects Agency Network*), precursora da actual Internet, tendo como principal objectivo o transporte, transparente para o utilizador, de informação através de um conjunto de nós e redes interligadas, com capacidade de reconfiguração dinâmica para lhe conferir robustez mesmo em presença de falhas. Nada fazia, no entanto, prever que esta rede com objectivos primordialmente militares e de investigação fosse evoluir para a Internet comercial que hoje se conhece, ou para a rede única com integração de uma multiplicidade de serviços que se pretende criar. Assim, não foi previsto o suporte nestes protocolos para serviços que não o transporte de dados com robustez mas sem garantias.

Com o objectivo de integrar o serviço de transporte de dados com outros serviços com requisitos de tempo-real têm surgido diversas tecnologias, representando diferentes abordagens ao mesmo problema. A tecnologia ATM, concebida desde o início como uma abordagem integrada ao suporte de serviços com diferentes requisitos e características numa única rede de telecomunicações, representa a solução mais completa para o problema da qualidade de serviço. Apresenta, no entanto, algumas limitações ao uso em larga escala, das quais a principal é a sua grande complexidade. Outras abordagens, como a de Serviços Integrados e a de Serviços Diferenciados, partem de um modelo Internet existente e implementado, tentando criar extensões que permitam obter qualidade de serviço sem obrigar à substituição de toda a infraestrutura de rede instalada.

Os principais objectivos do trabalho aqui apresentado são, por um lado, a análise de diversas tecnologias para implementação de qualidade de serviço, e por outro a construção de um ambiente de serviços diferenciados (*diffserv*) com base em *software* de código aberto a correr em *hardware* comum (computadores pessoais). Um terceiro objectivo é a implementação de um sistema de teste, baseado no mesmo tipo de ferramentas, e que permita avaliar o desempenho de um ambiente deste tipo.

Tanto os *routers diffserv* como o sistema de teste são baseados no sistema operativo Linux. As razões para a escolha deste sistema operativo são o facto de ter código aberto, de dispor dos mecanismos de controlo de tráfego necessários para suportar serviços diferenciados, de ter suporte para NAT (*Network Address Translation*), necessário no sistema de teste, e de existirem numerosas aplicações de geração e análise de tráfego que, por serem igualmente de código aberto, podem facilmente modificar-se por forma a adicionar toda a funcionalidade necessária. Em relação ao *hardware*, todas as máquinas são computadores pessoais com interfaces de rede *Fast Ethernet* e ATM, dispondo o sistema de teste de dois processadores em configuração SMP[1].

No segundo capítulo é feita uma análise, do ponto de vista arquitectónico, de diversas tecnologias disponíveis para a implementação de qualidade de serviço em redes de comutação de pacotes. Neste capítulo é inicialmente feita uma análise do conceito de qualidade de serviço e de como, sendo um conceito subjectivo, se pode caracterizar de forma objectiva através de um conjunto de parâmetros mensuráveis. De seguida é feita uma introdução genérica às possíveis aproximações para garantir qualidade de serviço. Por fim, são apresentadas diferentes arquitecturas que suportam ou visam a obtenção de qualidade de serviço, nomeadamente o ATM, o IEEE 802.1p, e os modelos Internet de Serviços Integrados (*intserv*) e de Serviços Diferenciados (*diffserv*), incluindo uma análise crítica das suas características, vantagens e inconvenientes, dos serviços que podem disponibilizar, e ainda da sua aplicabilidade no âmbito da Internet.

Em qualquer arquitectura de rede capaz de garantir qualidade de serviço é necessário existir um módulo, mais ou menos autónomo, de controlo de tráfego. É da responsabilidade deste módulo decidir a ordem e o instante de envio dos pacotes, isto é, efectuar o seu escalonamento. No terceiro capítulo são abordados os diversos algoritmos utilizados no controlo de tráfego em redes de comutação de pacotes, cada um com as suas caracteríticas próprias, ditadas normalmente pelo fim para o qual foram concebidos. Em particular, existem três objectivos distintos (e até contraditórios) para algoritmos de controlo de tráfego: a garantia de serviço, a distribuição equitativa de recursos e a diferenciação de serviços.

Entre os algoritmos de controlo de tráfego analisados contam-se alguns extremamente simples, como o FIFO ou o escalonador de prioridades estritas. Outros mais complexos pretendem aproximar um modelo fluido de tráfego apenas com as diferenças inerentes à existência de pacotes, como acontece com a família de algoritmos WFQ. Outros ainda, como o CSZ ou o CBQ, pretendem abordar de forma global e integrada os problemas dos diversos tipos de tráfego num *router*, ten-

[1] Multi-processamento simétrico.

tando satisfazer requisitos que à partida parecem contraditórios. Por fim, algoritmos como os da família RED, pretendem tirar partido de características dos protocolos (neste caso, do protocolo de transporte TCP) para conseguir uma gestão eficiente de recursos.

Relativamente à implementação de um ambiente de serviços diferenciados com base em equipamento genérico (computadores pessoais) e *software* de código aberto, neste caso o sistema operativo Linux, um dos problemas que cedo se manifestou foi a quase total inexistência de documentação. De facto, apesar de existir alguma documentação sobre a arquitectura do controlo de tráfego em Linux do ponto de vista do programador, a documentação a nível de utilizador para configuração das ferramentas disponíveis estava praticamente limitada a listas de correio electrónico (*linux-net* e *linux-diffserv*) e os respectivos arquivos. Para colmatar esta grande lacuna, o quarto capítulo foi dedicado na sua maior parte à documentação detalhada e exaustiva dos mecanismos de controlo de tráfego disponibilizados pelo sistema operativo Linux, com particular relevo à sua configuração, efectuada através da ferramenta `tc`.

Neste capítulo é ainda apresentado, sob os pontos de vista físico e lógico, o ambiente de desenvolvimento e teste implementado, cujos constituintes principais são o *router* onde são configurados os mecanismos de controlo de tráfego / QoS e o equipamento de teste propriamente dito. O equipamento de teste é particularmente importante, pois se ele próprio não for capaz de obter com precisão a informação necessária sobre o desempenho do sistema não é possível tirar conclusões credíveis. Em particular, no teste de qualidade de serviço em redes de comutação de pacotes é necessária uma precisão temporal difícil de obter sem utilizar equipamento especializado, o que contraria o propósito de o conseguir usando equipamento genérico. A solução encontrada foi a concentração do emissor e do receptor de tráfego numa única máquina com duplo processador e duas interfaces de rede. A nível de *software*, foi necessário recorrer a técnicas de NAT (*Network Address Translation*) para permitir a concentração das duas tarefas numa única máquina. Além disso, dada a inexistência de ferramentas com todas as características necessárias para a geração e recepção de tráfego, foram efectuadas as necessárias alterações ao pacote de *software* de código aberto escolhido para o efeito (`rude`) por forma a obter essas características.

O quinto capítulo é dedicado à configuração de serviços com base nas ferramentas anteriormente descritas, à apresentação dos testes efectuados ao seu desempenho e à discussão dos resultados obtidos. Os primeiros testes efectuados visam avaliar as próprias ferramentas usadas, por forma a conhecer as suas limitações, tendo-se verificado que é possível efectuar testes com resultados bastante precisos usando as ferramentas modificadas para o efeito. Foram também efectuados testes que conduziram à detecção de deficiências introduzidas em níveis inferiores ao de rede (ATM e AAL), e cuja origem e possíveis soluções são discutidas. Foram ainda efectuados testes a algoritmos específicos de controlo de tráfego por forma a avaliar a sua utilidade e as suas limitações na criação de serviços diferenciados. Por fim, apresentam-se *scripts* de configuração de nós *diffserv*, interior (*core*) e de fronteira (*edge*), suportando simultaneamente os serviços AF (*Assured Forwarding*), EF (*Expedited Forwarding*) e BE (serviço tipo melhor esforço), bem como os resultados de testes efectuados para a sua validação e avaliação do seu desempenho.

O sexto e último capítulo contém uma análise crítica do trabalho desenvolvido, com destaque para os resultados obtidos e as conclusões que dele se podem retirar. Apresentam-se ainda neste capítulo possíveis direcções a tomar para um desenvolvimento futuro do trabalho realizado.

As principais conclusões que podem retirar-se deste trabalho são, por um lado, a de que é possível configurar um ambiente de serviços diferenciados, bem como do equipamento de teste usado para avaliação do seu desempenho, com base inteiramente em *hardware* comum e ferramentas de *software* de código aberto, e por outro lado a de que para o conseguir com sucesso é necessário dar atenção aos pormenores de todas as camadas desde o nível físico ao nível de aplicação, pois só assim é possível obter verdadeiramente qualidade de serviço.

Capítulo 2

QoS e QoS em redes IP

Tradicionalmente as redes de comutação de pacotes, concebidas para a transferência de dados, disponibilizam apenas um serviço do tipo melhor esforço, em que todos os pacotes são tratados de igual modo[1]. No entanto, com a tendência para uma crescente integração de serviços na mesma rede e implantação de serviços multimédia distribuidos, a comutação de dados tipo melhor esforço não se tem mostrado adequada, por não suportar os novos serviços com a necessária qualidade.

A recomendação E.800 do ITU-T define Qualidade de Serviço (QoS) como:

> "O efeito colectivo do desempenho de um serviço que determina o grau de satisfação de um utilizador desse serviço."

Esta é uma definição subjectiva, uma vez que associa a qualidade de serviço à percepção que o utilizador tem desse serviço. Do ponto de vista da rede existem, no entanto, alguns parâmetros objectivos que podem quantificar a qualidade de um serviço fornecido pela rede, que serão apresentados de seguida.

2.1 Parâmetros objectivos de QoS

Atraso [2] é o intervalo de tempo que um pacote demora a atravessar a rede, desde o emissor até ao receptor. O atraso total é a soma de diversos componentes: atraso de transmissão — correspondente ao tempo necessário para colocar na rede o fluxo de *bits* que constituem o pacote; atraso de propagação — tempo que um bit demora a atravessar uma ligação física; atraso de processamento — tempo de processamento do pacote num nó (e.g., encaminhamento para a porta de saída); e o atraso em filas de espera. No nó de destino, existem atrasos adicionais, correspondentes à transferência do pacote no barramento, eventual cópia do núcleo (*kernel*) para espaço de utilizador, escalonamento da aplicação. Em algumas aplicações existe ainda um atraso adicional com a finalidade de absorver o *jitter*.

[1] O que, só por si, nem sequer garante que recebam um serviço equitativo.
[2] O atraso total entre o emissor e o receptor é também designado latência.

Jitter é a variação do atraso sofrido por um pacote na rede.

Débito é a taxa máxima de transferência de dados sustentável entre o emissor e o receptor. Esta taxa não depende apenas da infraestrutura física ao longo do percurso do tráfego, que apenas lhe impõe um limite superior, mas também de outros fluxos de tráfego que partilhem componentes do caminho entre um extremo e outro.

Fiabilidade é a capacidade de assegurar a entrega de pacotes no receptor tal como foram colocados na rede pelo emissor. Uma vez que as tecnologias de transmissão actuais tendem a ter taxas de erro bastante baixas, os principais factores que caracterizam a fiabilidade são a perda de pacotes e a sua entrega fora de ordem.

A implementação de mecanismos QoS visa dar determinadas garantias sobre estes parâmetros para que, independentemente da carga na rede, o serviço seja prestado de forma consistente e previsível, assegurando que o utilizador tenha dele uma percepção positiva.

Antes de passar à discussão dos mecanismos de QoS que permitem controlar os parâmetros mencionados, será feita uma breve análise de como esses parâmetros influenciam o desempenho dos serviços na rede.

Protocolos de transporte como o TCP, além de garantirem fiabilidade na entrega de dados ordenada e sem perdas, usam mecanismos de controlo de fluxo que permitem ao emissor adaptar o volume de dados transmitidos por unidade de tempo ao ritmo a que o receptor consegue processá-los e que a rede os consegue entregar. Estes mecanismos baseiam-se em informação transmitida em sentido inverso, contendo confirmações que avisam o emissor que a recepção dos dados foi bem sucedida.

Quanto maior for o atraso, maior será o volume de dados em trânsito na rede[3], e mais insensível se torna esta malha de realimentação, dificultando a acção do protocolo de transporte e tornando-o insensível a variações rápidas da carga na rede. No caso de aplicações de tempo real que usem protocolos de transporte não fiáveis como o UDP, valores elevados de atraso podem dificultar a interactividade, reduzir drasticamente a qualidade do serviço, ou mesmo torná-lo inútil.

Valores elevados de *jitter*, por seu lado, dificultam a acção do protocolo de transporte, levando-o a fazer do tempo de ida e volta (round trip time - RTT) uma estimativa conservadora. Isto vai originar a ocorrência de *timeouts* e, consequentemente, retransmissões eventualmente desnecessárias que tornam o protocolo ineficiente. Em aplicações de tempo real, na melhor das hipóteses o *jitter* é absorvido no *buffer* da aplicação no receptor, tendo um efeito semelhante ao aumento do atraso até ao seu valor máximo. Se o *buffer* não for capaz de absorver o *jitter*, os pacotes mais atrasados são considerados perdidos, o que provoca distorção do sinal, podendo mesmo levar à inutilização do serviço.

Quanto ao débito disponível, se o seu valor estiver abaixo dos requisitos mínimos da aplicação, torna-se impossível o seu uso.

[3]Até ao limite imposto pela janela de transmissão, no caso do TCP.

A falta de fiabilidade na rede afecta os serviços de diversas maneiras. Em aplicações que correm sobre um protocolo de transporte fiável, a perda de pacotes origina retransmissões que aumentam o atraso, além de despoletar mecanismos de controlo de congestionamento que provocam uma diminuição efectiva do débito. Em aplicações de tempo real, a perda de pacotes pode reflectir-se na distorção do sinal ou mesmo na impossibilidade de prestação do serviço. A entrega de pacotes fora de ordem, por seu lado, implica o consumo de recursos de memória e processamento no receptor para a sua ordenação, além de ter efeitos semelhantes a elevados valores de *jitter*[4].

Numa rede ideal todo o tráfego seria entregue com atraso nulo[5], tendo à sua disposição todo o débito necessário, e sem existência de perdas. Em redes reais, no entanto, os recursos são limitados. Consequentemente, alguns recursos da rede acabarão por manifestar-se insuficientes para satisfazer os requisitos das aplicações que sobre ela correm. Para garantir uma qualidade mínima a pelo menos alguns desses serviços, são necessárias medidas que serão discutidas na secção seguinte.

2.2 Aproximações para garantir QoS

A existência de uma rede "saudável"[6] é um pré-requisito para o fornecimento de serviços com qualidade. Esta afirmação pode parecer trivial, mas são inúmeros os casos de mau funcionamento de redes por subdimensionamento[7] ou má configuração dos protocolos de encaminhamento ou dos serviços básicos da rede (e.g., DNS). Por exemplo, uma má configuração dos protocolos de encaminhamento pode levar ao estabelecimento de laços de encaminhamento (*routing loops*), originando perda de pacotes ou mesmo isolamento de zonas da rede por periodos de tempo indeterminados.

Cumprido o requisito de a rede ser "saudável", existem dois tipos básicos de aproximação à qualidade de serviço:

- Sobredimensionamento
- Controlo de tráfego

A primeira aproximação consiste em dotar a rede de meios suficientes, nomeadamente no que diz respeito à capacidade das ligações físicas, à velocidade de processamento dos nós de encaminhamento e ao tamanho dos *buffers* nestes nós, para que todas as necessidades do tráfego que se prevê nela ir circular sejam satisfeitas, dispensando a existência de mecanismos explícitos de QoS. Esta aproximação pode ser válida no caso de pequenas redes locais em que não circule tráfego crítico de tempo real. Neste caso podemos admitir que na quase totalidade do tempo todos os parâmetros quantitativos de qualidade de serviço têm valores aceitáveis[8].

[4] De facto, a entrega fora de ordem pode ser vista como atraso de um pacote suficientemente grande para que seja entregue depois do seguinte, que tem um atraso menor.

[5] Ou apenas com o atraso de propagação. Em qualquer caso, sem *jitter*.

[6] Isto é, bem concebida, implementada e explorada

[7] Ou sobreexploração

[8] Esta realidade, no entanto, tende a ser temporária, uma vez que o volume de tráfego tende a crescer, eventualmente até um ponto que invalide o pressuposto de sobredimensionamento.

Esta aproximação, no entanto, não é válida para redes mais complexas, que podem abranger todo o planeta, como é o caso da Internet. Isto porque, por um lado os custos de o fazer seriam insustentavelmente elevados, e por outro rapidamente o tráfego oferecido à rede cresceria para valores excessivos para a capaciade da rede. É essencialmente para estes casos que foram desenvolvidos métodos de controlo de tráfego, baseados na diferenciação e fornecimento de diferentes níveis de serviço, que permitem à rede disponibilizar qualidade de serviço mesmo que o tráfego oferecido exceda as suas capacidades.

Podemos separar em dois modelos genéricos os diferentes métodos de controlo de tráfego:

- Baseados na reserva de recursos
- Sem reserva de recursos

No primeiro caso os recursos da rede são explicitamente identificados e reservados[9] recorrendo a protocolos de reserva dinâmica de recursos, bem como a mecanismos de controlo de admissão. Os nós da rede classificam os pacotes, baseando-se na informação das reservas efectuadas para diferenciar o seu tratamento.

No segundo caso não há reserva explícita de recursos. Em vez disso, cada pacote é associado a uma determinada classe. É com base no conjunto de classes, a que estão associados agregados de tráfego, que os nós da rede fazem a diferenciação de serviço. Note-se que mesmo assim pode ser necessário controlar o volume de tráfego de uma dada classe que pode ser injectado na rede por forma a garantir qualidade de serviço a outros pacotes da mesma classe que circulem na rede.

As aproximações à qualidade de serviço na rede podem ainda diferir quanto à camada protocolar em que actuam. Em relação ao modelo OSI de sete camadas, existem métodos que actuam sobre cada uma das três camadas inferiores.

1. Mecanismos ao nível físico

   A camada de nível físico ocupa-se da transmissão de *bits* entre dois nós da rede, dizendo respeito às especificações mecânicas, eléctricas e procedimentais das interfaces, bem como ao meio físico que as suporta. É, no entanto, possível actuar a este nível para obter diferenciação de níveis de serviço através da disponibilização para o tráfego de diversos caminhos com diferentes características, sendo a escolha efectuada consoante o nível de serviço desejado para cada pacote.

2. Mecanismos ao nível de ligação lógica

   É a este nível que actuam os mecanismos de qualidade de serviço em ATM, baseados no estabelecimento de circuitos virtuais (VC) com parâmetros de QoS associados. O ATM era visto até há pouco tempo como *a* solução de QoS. É ainda ao nível de ligação lógica que actua o mecanismo de prioridades IEEE 802.1p.

[9] Esta reserva pode ser estrita, com garantias rígidas, ou apenas probabilística.

3. Mecanismos ao nível de rede

   A base da Internet é a pilha protocolar TCP/IP, fazendo do protocolo de rede IP o denominador comum a toda a sua infraestrutura. Assim, esta é a camada de eleição para mecanismos que pretendam garantir qualidade de serviço extremo a extremo na rede global.

2.3 ATM

O ATM é uma tecnologia de transmissão e comutação de pacotes que surgiu da necessidade de integrar na mesma rede de comunicação diferentes serviços (áudio, vídeo e dados). Esta tecnologia foi adoptada pelo ITU-T como base para a implantação da RDIS de Banda Larga, mas a sua utilização estende-se também às redes locais[10].

A tecnologia ATM baseia-se na transmissão e comutação de pacotes de tamanho fixo, designados células. As células ATM têm um cabeçalho de 5 octetos e um campo de 48 octetos, designado *payload*, destinado ao transporte de dados. O tamanho das células é fixo por forma a permitir efectuar as funções de comutação e multiplexagem de forma rápida e eficiente, em *hardware*. O ATM é uma tecnologia orientada a conexões, pelo que é necessário establecer entre duas máquinas um canal virtual (VC) antes de iniciar a comunicação.

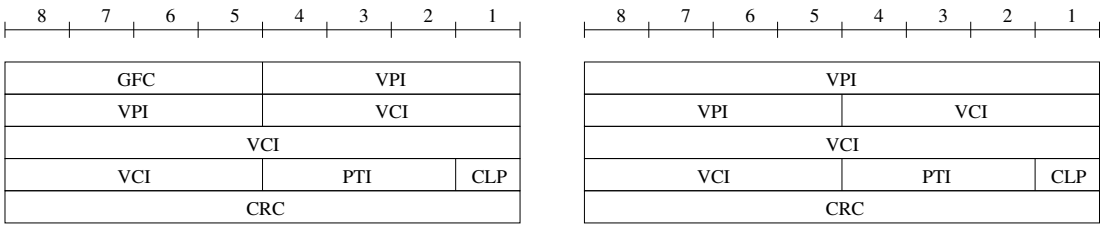

Figura 2.1: Cabeçalho das células ATM na UNI e na NNI.

Um canal virtual (VC) é identificado por dois campos do cabeçalho ATM: o identificador de caminho virtual (VPI) e o identificador de canal virtual (VCI). Um caminho virtual pode conter diversos canais virtuais. O equipamento terminal faz terminação de VC, mas os comutadores podem fazer apenas terminação de VP, tratando todo o tráfego dos VC nele contidos como um único fluxo, optimizando o processo de encaminhamento.

O campo PTI permite distinguir o tráfego de administração e gestão (OAM) do tráfego do utilizador. O campo de um único bit CLP identifica as células que devem ser prioritariamente eliminadas. O CRC, calculado apenas em relação ao cabeçalho da célula, permite verificar a ocorrência de erros. Deve ser calculado e verificado em todos os sistemas que terminam a camada ATM.

Na fase de estabelecimento de um circuito virtual são negociados parâmetros de tráfego e de QoS.

[10] De facto, o ATM popularizou-se em ambientes LAN mesmo antes da sua adopção em larga escala pelos operadores de telecomunicações.

2.3.1 Funções de controlo de tráfego

Concebido desde o início com vista à integração de serviços com garantias de QoS, o ATM dispõe de um conjunto bastante completo de funções de controlo de tráfego que permitem a validação do tráfego segundo as características de tráfego e QoS negociadas. Essas funções são:

Connection Admission Control (CAC): O controlo de admissão de conexões decide se uma conexão é ou não aceite conforme os recursos na rede sejam ou não suficientes para a suportar com a qualidade de serviço requerida e sem alterar as garantias dadas às conexões existentes.

Feedback Controls: Conjunto de acções tomadas pela rede e pelos equipamentos terminais por forma a regular o tráfego nas conexões ATM de acordo com o estado dos elementos da rede.

Usage Parameter Control (UPC): O controlo de parâmetros de utilização, também designado policiamento de tráfego, permite a monitorização e controlo das conexões ATM na interface UNI por forma a garantir que os contratos negociados durante o estabelecimento do VC são cumpridos por parte do utilizador. Caso contrário, a rede pode marcar ou eliminar células, recorrendo a algoritmos da família do *Leaky Bucket*.

Network Parameter Control (NPC): Semelhante ao UPC mas para interfaces NNI.

Cell Loss Priority Control: Em algumas categorias de serviço as células podem ser enviadas com o campo CLP do cabeçalho activado. Este campo de 1 bit do cabeçalho ATM permite identificar as células menos importantes de um fluxo. Caso se torne necessário a rede eliminar células, o facto de eliminar preferencialmente aquelas marcadas com o campo CLP a 1 permite que seja respeitada ao máximo a qualidade de serviço para o tráfego prioritário.

Explicit Forward Congestion Indication: Permite aos elementos da rede anunciar, através do campo PTI do cabeçalho ATM, a ocorrência de congestionamento ao longo do caminho.

Selective Cell Discard: A eliminação selectiva de células significa que em caso de congestionamento são eliminadas preferencialmente as células marcadas com o campo CLP do cabeçalho activo.

Traffic Shaping: A formatação de tráfego é um mecanismo que modifica as características de um fluxo de tráfego através do espaçamento adequado entre células e/ou limitação do débito máximo por forma a garantir que são respeitados determinados parâmetros de tráfego.

Network Resource Management (NRM): Gestão de recursos da rede efectuada através dos caminhos virtuais (VP) por forma a optimizar a reserva de recursos e minimizar a probabilidade de congestionamento.

Frame Discard: Em caso de congestionamento a rede pode eliminar selectivamente tramas de nível superior em lugar de simples células.

Generic Flow Control (GFC): Usado na UNI para fins de controlo de fluxo.

ABR Flow Control: Protocolo usado para controlo de fluxo ABR[11] para a partilha do débito disponível pelos diversos fluxos com esta categoria de serviço de forma adaptativa.

Forward Error Correction (FEC): Mecanismo que permite a recuperação de tramas após perda de células.

2.3.2 Parâmetros de tráfego

Durante o estabelecimento da conexão o utilizador negocia com a rede um contrato onde são especificados parâmetros de tráfego e de qualidade de serviço. Os parâmetros de tráfego, usados pela função UPC da rede para verificação de conformidade do fluxo de células, são os seguintes:

Peak Cell Rate (PCR): É um limite superior ao débito de células que a fonte não pode exceder. Dito de outra forma, é o inverso do tempo mínimo entre células. Note-se que este valor se refere à camada ATM[12] e não ao fluxo multiplexado de células, pelo que do ponto de vista prático pode entender-se como o limite superior para o débito médio durante uma rajada (*burst*) de células.

Sustainable Cell Rate (SCR): Limite superior para o débito médio de um fluxo conforme. No tráfego CBR, este valor será igual ao PCR, mas em tráfego de outras categorias será inferior, reflectindo o débito médio necessário à aplicação.

Minimum Cell Rate (MCR): Na categoria de tráfego ABR, o MCR representa o débito mínimo garantido. A aplicação poderá sempre enviar a um débito maior ou igual a este valor[13].

Burst Tolerance (BT): Parâmetro usado em serviços VBR. O seu valor obtém-se pela seguinte fórmula:

$$BT = (MBS - 1)\left(\frac{1}{SCR} - \frac{1}{PCR}\right)$$

Maximum Burst Size (MBS): Nas mensagens de sinalização o parâmetro BT é expresso através do MBS, que representa o número máximo de células consecutivas enviadas com um débito igual ao PCR (rajada). Para o SCR não ser excedido, deve controlar-se o intervalo entre rajadas.

Cell Delay Variation Tolerance (CDVT): Especifica a variação em relação ao valor nominal do instante em que as células são injectadas na rede (*jitter*) que a rede pode suportar.

[11] Categoria de serviço descrita na secção 2.3.4.
[12] Estruturação por camadas e planos do ATM disponível em [1].
[13] Se a aplicação não necessitar de um débito mínimo garantido pode dar um valor nulo a este parâmetro.

2.3.3 Parâmetros de QoS

Se os parâmetros de tráfego especificam limites impostos à fonte para o tráfego que injecta na rede, os parâmetros de QoS exprimem as obrigações da rede no que respeita à qualidade de serviço entregue. São eles os seguintes:

Cell Loss Ratio (CLR): Exprime a relação entre o número de células que podem perder-se na rede e o número total de células do fluxo injectadas na rede.

Cell Error Ratio (CER): Exprime a relação entre o número de células que podem conter erros e o número total de células transferidas.

Severely Errored Cell Block Ratio (SECBR): Exprime a relação entre o número de blocos severamente corrompidos e o número total de blocos. Um bloco considera-se severamente corrompido se excede um dado nível de células com erros, células perdidas ou células mal inseridas.

Cell Misinsertion Rate (CMR): É a taxa de células inseridas num fluxo que pertencem a outro, ou seja, incorrectamente encaminhadas.

Cell Transfer Delay (CTD): Exprime o tempo decorrido entre a injecção pela fonte de uma célula na rede até que esta atinge o seu destino, incluindo o tempo de processamento e em filas de espera nos comutadores e o tempo de transmissão entre eles.

Cell Delay Variation (CDV): Exprime a diferença entre os valores máximo e mínimo do CTD.

Destes parâmetros, os mais importantes são CLR, CTD e CDV, uma vez que a forma como são controlados vai determinar as diferentes categorias de serviço ATM.

2.3.4 Categorias de serviço

Na fase de estabelecimento de uma conexão o utilizador deve especificar a categoria de serviço pretendida. Esta é determinada pela classe de serviço, e vai ter implicações ao nível do uso dos parâmetros de tráfego e QoS.

O ATM Forum definiu as seguintes categorias de serviço:

Constant Bit Rate (CBR): Destina-se a aplicações de tempo real de débito constante e com apertados requisitos relativamente a perdas e a atrasos. O débito é caracterizada pelo parâmetro PCR, ficando reservada, pelo que mesmo que a aplicação transmita com débito inferior, não pode ser usada por outro tráfego[14]. Dadas as suas características, esta categoria de serviço é apropriada para emulação de circuitos, vídeo conferência e transporte de voz, áudio e vídeo com débito fixo.

[14]Na realidade os *slots* de tempo destinados a um fluxo CBR e não utilizados podem ser usados para transportar tráfego UBR, mas para efeitos de admissão de chamadas (CAC) não é possível fazer qualquer tipo de multiplexagem estatística.

Real-Time Variable Bit Rate (rt-VBR): Destina-se a aplicações de tempo real com débito variável, mas com forte dependência temporal entre a fonte e o destino. Um VC de categoria rt-VBR é caracterizado pelos valores de PCR, SCR e MBS. Uma vez que este tipo de aplicações é sensível ao atraso, a rede deve ainda respeitar o limite superior do atraso (CTD) e de variação do atraso (CDV). Esta categoria destina-se essencialmente a aplicações multimédia que façam uso de algoritmos com perdas (*lossy*), de débito variável, e que possam suportar a perda de um pequeno número de células sem sacrificar demasiado o desempenho do serviço. Exemplos deste tipo de aplicação são o transporte de áudio e vídeo comprimidos.

Non-Real-Time Variable Bit Rate (nrt-VBR): Esta categoria é semelhante à anterior em todos os aspectos excepto nas garantias quanto ao atraso, que neste caso são inexistentes. É adequada ao transporte de áudio ou vídeo não interactivos.

Unspecified Bit Rate (UBR): Nesta categoria de serviço a rede não oferece qualquer garantia, pelo que se adequa apenas a tráfego tipo melhor esforço, bastante tolerante ao atraso e a perdas. Exemplos de aplicações adequadas a esta categoria são a transferência de ficheiros e o correio electrónico.

Available Bit Rate (ABR): Nesta categoria são negociados o MCR e o PCR. Não são dadas quaisquer garantias quanto ao atraso, e quanto ao débito apenas se garante o MCR. Quanto a perdas, a rede garante que o seu valor se mantém baixo desde que a fonte adapte o seu débito conforme a informação de controlo que a rede emite em sentido inverso sob a forma de células RM. É geralmente considerada a categoria de eleição para transportar tráfego TCP/IP.

2.3.5 Prós e contras do ATM

O ATM fornece um conjunto bastante completo de mecanismos de QoS que actuam ao nível da ligação lógica, baseados em reserva por meio de sinalização, que lhe permitem disponibilizar um serviço previsível, com garantias rígidas quanto aos parâmetros que quantificam a qualidade de serviço: atraso, *jitter*, débito e fiabilidade. No entanto, isto tem um preço: com os mecanismos de sinalização e a necessidade de manter informação de estado por VC em cada nó, o ATM é uma tecnologia complexa.

Outro problema do ATM é a falta de aplicações que o suportem nativamente para poder tirar partido das garantias de QoS que oferece. De facto, à excepção de algumas aplicações desenvolvidas por instituições académicas ou de investigação, constata-se a quase total inexistência de aplicações que utilizem nativamente o ATM. Assim, a maior parte do tráfego que acaba por circular em redes ATM é tráfego TCP/IP, a ubíqua pilha protocolar da Internet, usando as categorias de serviço UBR ou ABR, deixando por explorar as possibilidades de QoS.

Um outro problema é que para explorar as capacidades de QoS do ATM é necessária a existência deste de um extremo ao outro da rede. Uma vez que a maior parte do tráfego é transportado por protocolos de nível superior, nomeadamente TCP/IP, pode não existir ATM ao longo da totalidade do percurso do tráfego. Deste modo, vão ser introduzidos atrasos e perdas nos *routers* dos troços

não-ATM. Assim, embora o ATM possa ser usado para garantir QoS a nível de uma organização ou de um ISP, dificilmente o seu uso se poderia estender à totalidade da Internet.

2.4 IEEE 802.1p

A recente introdução pelo IEEE 802.1 Internetworking Task Group do 802.1p como parte da revisão à norma 802.1D, deu às LAN comutadas um impulso em direcção à convergência de voz, áudio, vídeo e dados.

A especificação 802.1p introduz nas redes IEEE 802 um mecanismo de prioridade de tráfego para melhorar o suporte de tráfego crítico, e a filtragem dinâmica de tráfego *multicast*, que permite limitar o volume deste tipo de tráfego em LAN comutadas[15]. Embora ambos sejam interessantes do ponto de vista do desempenho, o mecanismo de prioridades tem um impacto mais directo na qualidade de serviço das redes IEEE 802, pelo menos a curto prazo.

O formato das tramas IEEE 802.3 (*ethernet*) não inclui nenhum campo que possa transportar a informação de prioridade, pelo que neste tipo de redes deve ser usado o formato definido pela norma IEEE 802.1Q, concebido principalmente para a identificação de VLAN, mas que inclui um campo de três *bits*, designado *user_priority*, destinado a transportar essa informação. Note-se que isto pode implicar a substituição de equipamento antigo por novo equipamento que suporte este formato.

O mecanismo de prioridades 802.1p actua ao nível de ligação lógica (nível 2) do modelo de referência OSI. Para dele fazerem uso, os comutadores devem ter a capacidade de mapear o tráfego em diferentes classes. A norma 802.1p define oito valores de prioridade, deixando ao gestor da rede a atribuição desses valores aos diferentes tipos de tráfego na rede. Existem, no entanto, directivas sobre a forma como fazer esse mapeamento: com prioridade 7, a mais elevada, deve ser transportado tráfego crítico para a rede, tal como actualizações de rotas[16]; as prioridades 5 e 6 são indicadas para aplicações sensíveis ao atraso, como o transporte de voz, áudio e vídeo interactivos; as classes de 4 a 1 podem transportar desde tráfego com carga controlada[17], tal como áudio ou vídeo não interactivos (classe 4) até tráfego de baixa prioridade pouco afectado por perdas (classe 1); a classe 0 tem um significado especial, sendo usada quando não foi especificada uma classe em particular, e tem um comportamento do tipo melhor esforço.

Os equipamentos[18] não têm obrigatoriamente que implementar oito diferentes filas de espera por porta, uma por valor de prioridade. Na tabela 2.1 mostra-se o mapeamento recomendado entre o valor de prioridade de uma trama e a fila de espera em que deve ser colocada[19], conforme o número de filas de que o equipamento disponha.

É de salientar que, ao contrário do que seria de esperar, tramas com valor 0 no campo *user_priority*

[15] ou *bridged*

[16] Protocolos de sincronização temporal, como o NTP, podem também beneficiar da atribuição deste nível de prioridade

[17] *Controlled load*

[18] Nomeadamente comutadores

[19] Considerando que um número de fila maior corresponde a um tratamento preferencial, i.e., maior prioridade

		Nº de filas disponíveis							
		1	2	3	4	5	6	7	8
Prioridade	0	0	0	0	1	1	1	1	2
	1	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0	1
	3	0	0	0	1	1	2	2	3
	4	0	1	1	2	2	3	3	4
	5	0	1	1	2	3	4	4	5
	6	0	1	2	3	4	5	5	6
	7	0	1	2	3	4	5	6	7

Tabela 2.1: Mapeamento entre valor de prioridade e fila de espera para diferentes números de filas de espera

não têm prioridade inferior àquelas que contêm o valor 1 neste campo. Isto acontece porque, como foi anteriormente referido, o valor 0 tem significado especial. No mapeamento recomendado a prioridade deste tráfego está entre a dos níveis 2 e 3, implicando que tráfego com *user_priority* 1 ou 2 tem um comportamento pior que tráfego genérico, tipo melhor esforço, sem uma prioridade em particular atribuída.

Outro aspecto que convém referir é que apesar de o escalonador definido como padrão na norma 802.1D ser de prioridade estrita, é possível que algum equipamento implemente algoritmos mais complexos, baseados noutros aspectos além da prioridade. Neste caso é provável que se torne conveniente redefinir a tabela de mapeamentos, substituindo-a por uma mais apropriada.

Apesar de especificar um mecanismo de diferenciação de tráfego capaz de reordenar tramas por forma a garantir a entrega prioritária de tráfego crítico, o 802.1p só por si não pode dar qualquer garantia quanto à latência, o que o torna inadequado para aplicações que necessitem de garantias rígidas quanto a este parâmetro. Para aplicações deste tipo uma tecnologia como o ATM é muito mais apropriada. No entanto, se a simples diferenciação for suficiente, ou se usado conjuntamente com outros mecanismos de QoS, nomeadamente de nível 3, o 802.1p pode ser vital na integração de tráfego de diferentes serviços na mesma rede.

2.5 Serviços Integrados e RSVP

Com o objectivo de integração de serviços na Internet, o IETF formou um grupo de trabalho, o *Integrated Services*[20] *Working Group*, que definiu classes de serviço que, se suportadas por todos os *routers* ao longo do percurso de um fluxo de dados, permitem dar certas garantias de QoS ao referido fluxo. O tipo de garantia e a sua quantificação são configuráveis por fluxo, de acordo com as necessidades da aplicação. Os requisitos da aplicação podem ser comunicados aos *routers* através de procedimentos de gestão, mas o método mais comum é o uso de um protocolo de reserva como o

[20] Ou, abreviadamente, *intserv*.

RSVP, que será descrito mais adiante. Desta forma os *routers* podem reservar recursos[21] e adaptar o escalonamento de pacotes de modo a cumprir as garantias dadas ao fluxo quanto à qualidade de serviço extremo a extremo.

Por forma a determinar exactamente que recursos têm que ser reservados é necessário que o *router* tenha em conta os mecanismos de QoS disponibilizados pela camada de ligação lógica. Se esta disponibilizar mecanismos de QoS configuráveis através de negociação, é da responsabilidade do *router* efectuar essa negociação uma vez aceite o pedido de QoS para o fluxo. Isto implica um mapeamento das garantias atribuídas ao fluxo em parâmetros de QoS ao nível 2, tendo sido designado pelo IETF um outro grupo de trabalho, o *Integrated Services over Specific Lower Layers*[22], para definir os mapeamentos para as diferentes tecnologias existentes a este nível. No caso particular de a camada de ligação lógica não suportar qualquer mecanismo de QoS, como acontece em linhas dedicadas, não é necessário esse mapeamento, uma vez que é o próprio *router* a controlar inteiramente o tráfego, sendo esse controlo efectuado ao nível 3.

A arquitectura de Serviços Integrados preconiza a existência de um controlo de admissão ao qual or *routers* devem submeter todos os pedidos de QoS, por forma a assegurar que apenas são aceites aqueles que os recursos disponíveis permitam suportar, tendo em conta os parâmetros que caracterizam o tráfego no fluxo. Em particular, apenas podem ser aceites fluxos cujo tamanho máximo de pacote não exceda a MTU[23] da ligação, uma vez que o modelo de Serviços Integrados assume a ausência de fragmentação de pacotes em fluxos com QoS.

Uma vez efectuada a reserva de recursos ao longo de todo o percurso do fluxo de tráfego, o fluxo pode esperar da rede a qualidade de serviço negociada, desde que não ocorram modificações no seu percurso[24] e o tráfego seja conforme aos parâmetros anunciados. Por forma a evitar que o tráfego de fluxos não conformes influencie negativamente o desempenho de outros fluxos e/ou do tráfego com serviço tipo melhor esforço, estão previstos a formatação (*shaping*) e o policiamento de tráfego, de acordo com a classe de serviço. Se o fluxo não for conforme, o *router* pode tomar uma das seguintes acções:

- Dividir os pacotes em dois grupos, conformes e não conformes, dando a estes últimos um tratamento de melhor esforço.
- Degradar o serviço a todos os pacotes do fluxo de forma uniforme.
- Eliminar os pacotes não conformes do fluxo.

Embora o primeiro caso seja o mais usual, o tratamento dado a fluxos não conformes deve ser configurável, uma vez que há situações em que esta opção não é a mais adequada.

[21] Débito, espaço em *buffers*, etc.
[22] Abreviadamente *issll*.
[23] Unidade máxima de transferência.
[24] Isto porque o encaminhamento pode ser independente da sinalização/configuração de QoS. No entanto, se for usado um protocolo de sinalização QoS *soft-state* com refrescamento periódico, como acontece com o RSVP, pode ser possível recuperar automaticamente, embora com uma degradação temporária da qualidade de serviço. Se o protocolo de encaminhamento levar em conta a qualidade de serviço, este problema não existe.

A arquitectura *intserv* não se limita a permitir reservas para tráfego *unicast*. De facto, ela foi concebida tendo em mente a existência de serviços *multicast*, entre as quais áudio- e vídeo-conferência, estando previstos mecanismos de separação e agregação de fluxos[25].

Das classes de serviço inicialmente consideradas pelo IETF, apenas duas foram formalmente especificadas para uso com RSVP[26] na implementação de referência, o Serviço de Carga Controlada (*Controlled-Load Service*) e o Serviço Garantido (*Guaranteed Service*).

2.5.1 Serviço de Carga Controlada

O serviço de carga controlada emula o funcionamento de um serviço de melhor esforço atravessando o mesmo percurso com a rede submetida a baixos níveis de carga. A grande diferença entre um serviço deste tipo e um de tipo melhor esforço é que um fluxo em carga controlada, desde que conforme, não sofre uma deterioração significativa do serviço com o aumento de carga na rede. Assim, um fluxo em carga controlada terá consistentemente uma percentagem de pacotes perdidos pouco superior à introduzida pela taxa de erros do próprio meio físico, bem como um atraso que na quase totalidade dos pacotes é pouco superior ao atraso mínimo possível para um pacote nesse percurso[27].

Os *routers* ao longo do percurso seguido pelo fluxo devem certificar-se de que dispõem de recursos suficientes para o suportar. Para tal, devem submeter a controlo de admissão o pedido de QoS, que inclui uma parametrização do tráfego que irá circular no fluxo, *TSpec*, que contém os seguintes campos:

p	débito de pico no fluxo (octetos/segundo)
r	débito do *token bucket*[28] (octetos/segundo)
b	profundidade do *token bucket* (octetos)
m	unidade mínima de policiamento[29] (octetos)
M	tamanho máximo do datagrama (octetos)

O parâmetro r é um majorante do débito médio pois define, juntamente com o parâmetro b, um *token bucket* que o tráfego deve respeitar. O parâmetro p tem pouco significado no caso do serviço de carga controlada, podendo os *routers* desprezá-lo.

Este tipo de serviço é adequado para aplicações que suportem algumas perdas e atraso, desde que sejam limitados a níveis relativamente baixos. O serviço de carga controlada é apropriado para aplicações multimédia adaptativas de tempo real, como áudio- e vídeo-conferência. O transporte de

[25] Para mais informação detalhada sobre estes mecanismos nos serviços de carga controlada e garantido consultar [2] e [3].

[26] Vêr secção 2.5.4.

[27] Que corresponde à soma do atraso de propagação no meio com o atraso de processamento nos nós de encaminhamento.

[28] Débito de geração de *tokens*, correspondente ao débito médio controlado pelo *token bucket*.

[29] Pacotes de tamanho inferior são considerados deste tamanho para efeitos de policiamento.

protocolos como o SNA e o DEC LAT, sensíveis a atrasos e perdas, em túneis através da Internet é um outro exemplo de aplicação que pode beneficiar do serviço de carga controlada. Este serviço não é adequado, no entanto, para aplicações que exijam garantias rígidas quanto ao atraso ou perdas introduzidas pela rede.

2.5.2 Serviço Garantido

Ao contrário do serviço de carga controlada, o serviço garantido assegura a fluxos conformes um certo débito, um limite firme do atraso introduzido pela rede e a ausência de perdas nas filas de espera, desde que o tráfego seja conforme. Isto consegue-se usando uma aproximação do modelo fluido de serviço, sendo efectuada uma reserva baseada no *TSpec*, mencionado na secção anterior, e no *RSpec*, descrito adiante.

Em cada *router* ao longo do percurso seguido pelo fluxo é efectuada a reserva de um certo débito, R, e espaço em *buffer*, B. O modelo fluido garante que um fluxo que respeite um *token bucket* de débito r e profundidade b sofre, nestas condições, um atraso em fila de espera não maior que b/R desde que R seja superior a r. No entanto, este modelo apenas é aproximado, uma vez que não se trata de uma linha dedicada, mas sim de uma parcela do débito total, partilhado com outros fluxos. Por este motivo é necessário introduzir dois parâmetros de erro, C (dependente de R) e D (independente de R), que caracterizam o desvio máximo em relação ao modelo fluido perfeito, passando o limite do atraso a ser $b/R + C/R + D$. No entanto, o débito de pico do fluxo está limitado ao valor p e o tamanho de pacote ao valor M do *TSpec*. Consequentemente, o limite ao atraso extremo-a-extremo em filas de espera pode ser calculado com maior precisão recorrendo à seguinte fórmula [4]:

$$Atraso = \begin{cases} \frac{(b-M)(p-R)}{R(p-r)} + \frac{(M+C_{tot})}{R} + D_{tot} & se \quad p > R \geq r \\ \\ \frac{(M+C_{tot})}{R} + D_{tot} & se \quad R \geq p \geq r \end{cases} \tag{2.1}$$

onde C_{tot} e D_{tot} correspondem à soma dos parâmetros C e D de todos os *routers* ao longo do percurso. Note-se que no segundo caso, como o débito de pico é inferior ao reservado não existe a parcela de atraso devida ao esvaziamento do *token bucket*, dependente da sua profundidade, b.

A especificação de reserva, *RSpec*, além de um valor de débito, R (octetos/segundo), inclui um valor S (milissegundos) de folga[30], significando a diferença entre o atraso máximo desejado pela aplicação e o atraso total calculado pela fórmula anteriores, que pode ser usado pelos *routers* para flexibilizar a reserva de recursos, como se verá adiante. O espaço em *buffer* a reservar pelo *router* é calculado a partir de *TSpec* e *RSpec*. Note-se que o facto de haver um valor de débito reservado para um fluxo não significa que seja desperdiçado se este não o usar. De facto, é possível e recomendado usá-lo para, temporariamente, beneficiar outros fluxos.

As reservas de QoS de um fluxo podem não ser estáticas, mas sempre que o fluxo queira

[30] *Slack*.

modificar a reserva que lhe está atribuida, a nova reserva deve ser igualmente submetida a controlo de admissão. No entanto, se a nova reserva for inferior à existente[31], deverá ser sempre aceite.

O tráfego em fluxos com serviço garantido tem que ser policiado nos nós de acesso para garantir conformidade com a especificação *TSpec*. No interior da rede não é feito policiamento, mas sim reformatação do tráfego, nomeadamente nos nós em que há separação[32] ou agregação de fluxos. A reformatação é baseada na combinação de um *buffer* com um *token bucket* e um regulador de débito de pico. Se o *buffer* de reformatação encher, significa que o fluxo não é conforme, devendo os pacotes excedentes ter tratamento do tipo melhor esforço[33]. Deste modo, o mecanismo de reformatação efectua, ele próprio, policiamento.

O Serviço Garantido permite suportar aplicações com requisitos estritos de limitação do atraso, bem como da inexistência de perdas em filas de espera. É, portanto, adequado à emulação de circuitos e todas as suas aplicações.

2.5.3 Implementação de referência

A implementação de referência da arquitectura de serviços integrados inclui as funções necessárias ao fornecimento dos serviços descritos, incluindo o suporte de QoS disponibilizado pelas camadas inferiores, funções de controlo de tráfego, o protocolo de reserva RSVP[34] e policiamento/reformatação.

A figura 2.2 mostra a implementação de referência do modelo de Serviços Integrados, estruturada em blocos.

O classificador de pacotes, o escalonador e o módulo de controlo de admissão, em conjunto, formam o bloco de controlo de tráfego, que permite implementar os serviços anteriormente descritos.

O classificador de pacotes permite identificar, além do destino do pacote para efeitos de encaminhamento, a que fluxo pertence e a classe de QoS que lhe está atribuída. O escalonador gere as filas de espera e a transmissão de pacotes, devendo também negociar os parâmetros de QoS da camada inferior se esta o suportar.

O módulo de controlo de admissão determina a existência de recursos, como já foi referido. O módulo de controlo administrativo (*policy control*) determina se existe permissão para atribuir a classe de QoS desejada ao fluxo, possivelmente através da consulta a servidores externos (*policy servers*).

Nos equipamentos terminais existe um *daemon* RSVP cujos serviços são invocados pela aplicação através de uma API, e que troca mensagens de sinalização com o processo RSVP existente nos *routers*. Além da sinalização relativa à qualidade de serviço, da manutenção de informação de estado dos fluxos e da comunicação com o bloco de controlo de tráfego, é ainda da responsabilidade do processo RSVP a comunicação com o módulo de controlo administrativo para autenticação e autorização dos pedidos de QoS[35].

[31] Nos termos de [3].

[32] Quando há separação de fluxos, apenas é necessária formatação se os fluxos divergentes tiverem diferentes características.

[33] Ou outro, conforme a acção configurada para tráfego não conforme.

[34] Descrito na secção 2.5.4.

[35] Para controlo administrativo pode usar-se um protocolo como o COPS [5], cuja integração com o RSVP se

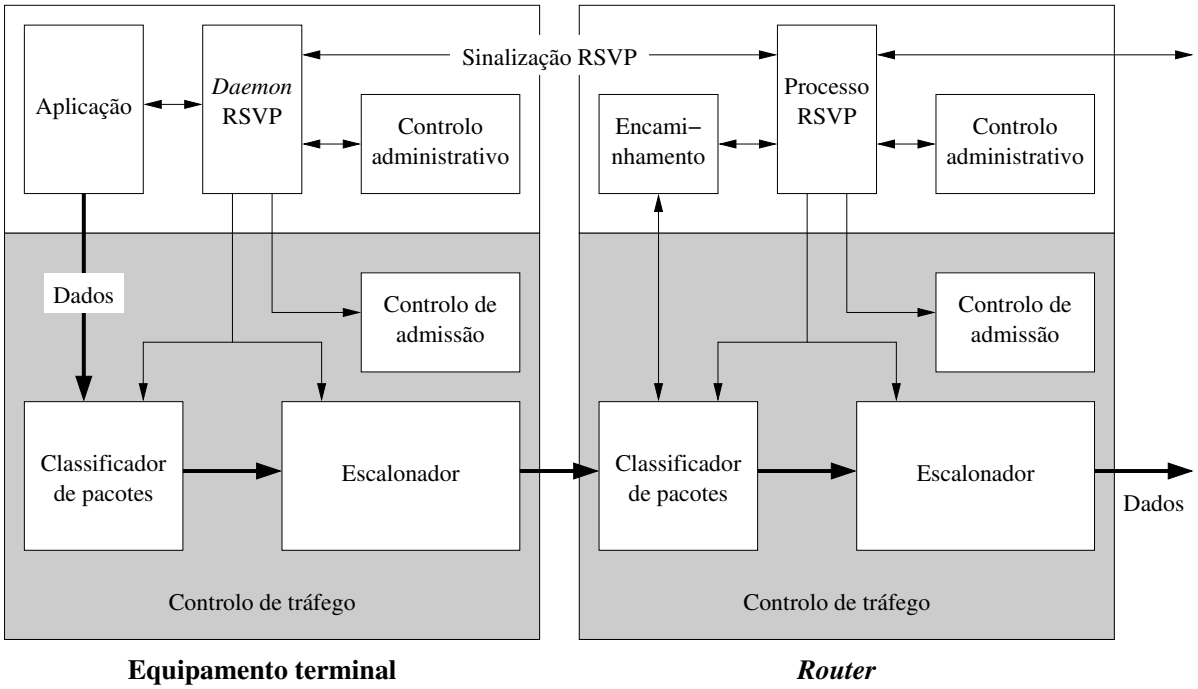

Figura 2.2: Implementação de referência dos Serviços Integrados

2.5.4 RSVP

Embora o modelo de serviços integrados seja independente do protocolo de reserva, podendo inclusivamente as reservas ser efectuadas através de um protocolo de gestão ou até por configuração estática, a sua implementação baseia-se, normalmente, no uso do protocolo RSVP (*Resource reSerVation Protocol*)[36].

Antes de mais, convém deixar claro que o RSVP não é um protocolo de encaminhamento, mas sim um protocolo de sinalização que opera através de rotas, *unicast* ou *multicast*, estabelecidas *a priori* pelo protocolo de encaminhamento existente na rede.

Algumas particularidades do RSVP são:

- Reservas unidireccionais (*simplex*)
- Reservas efectuadas pelo receptor
- Agregação de reservas numa árvore *multicast*
- Volatilidade das reservas (*soft-state*)
- Alteração dinâmica de reservas

discute em [6].

[36]Existem outros protocolos concebidos para este efeito, mas o RSVP, pelas suas características, é usado quase exclusivamente.

O RSVP foi concebido tendo em mente não só aplicações *unicast*, mas também aplicações *multicast* em que pode haver um grande número de receptores passivos. Assim, faz sentido que as reservas sejam unidireccionais. Se for necessária a bidireccionalidade, esta pode obter-se efectuando duas reservas em sentidos opostos, com características que podem ser ou não simétricas.

Embora pudesse parecer mais natural a reserva ser efectuada pelo emissor, uma vez que este tem a maior quantidade de informação sobre as características do fluxo, é preciso não esquecer que o RSVP foi concebido também para aplicações *multicast*, em que pode existir um número potencialmente elevado de receptores, tornando-se impossível ou, pelo menos, ineficiente, manter no emissor informação sobre todos eles. As reservas iniciadas pelo receptor evitam este problema. Além disso, como se verá adiante, a especificação *TSpec* do tráfego no fluxo é incluída nas mensagens *Path*, periodicamente enviadas pela fonte, pelo que os *routers* têm acesso a essa informação.

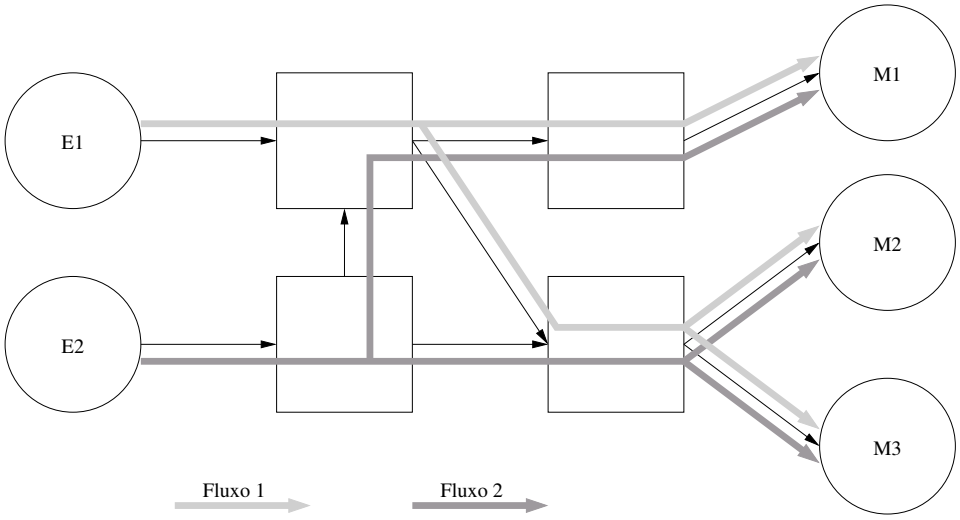

Figura 2.3: Sessão RSVP com fluxos *multicast*.

Por outro lado, para que fluxos *multicast* com um elevado número de utilizadores não sobrecarreguem desnecessariamente a rede, o RSVP permite a agregação de reservas, ainda que os parâmetros do serviço não sejam idênticos para todas elas. O pedido de QoS à saída do *router*[37] deve ter características que permitam satisfazer todos os pedidos à entrada. A figura 2.3 mostra um exemplo de sessão *multicast* com agregação de reservas.

Outra particularidade do RSVP é que as reservas são voláteis[38]. Isto acontece para manter tanto quanto possível a robustez inerente a uma Internet com serviço de melhor esforço, que em caso de falha de um nó, ou ligação entre um par de nós, rapidamente recupera o bom funcionamento alterando as rotas sem qualquer intervenção do utilizador. Para tal, as reservas têm que ser periodicamente refrescadas, pelo que mesmo que a rota se altere não é necessário terminar a reserva antiga e efectuar nova reserva. Além disso, as reservas podem ser alteradas dinamicamente,

[37] Em direcção à fonte.
[38] *Soft-state*.

bastando alterar a reserva no pedido de refrescamento. Naturalmente, a nova reserva está sujeita a controlo de admissão, pelo que pode falhar se a alteração for no sentido de aumentar os requisitos.

Uma sessão RSVP é identificada por um endereço de destino a nível de transporte, constituído por um endereço IP, um identificador de protocolo de transporte e, eventualmente, um identificador da porta (*port*), específico do protocolo de transporte. É de salientar que o endereço IP pode ser um endereço *multicast*.

Na mesma sessão RSVP podem existir vários emissores, podendo a reserva abranger os pacotes de todos eles ou apenas de um subconjunto. A especificação que identifica quais os emissores abrangidos designa-se *filter spec*, e é usada pelo classificador de pacotes (ver figura 2.2) para mapear os pacotes apropriados na respectiva classe. A especificação da quantidade de recursos reservados, por seu lado, designa-se *flowspec*, e é usada para parametrizar essa classe no escalonador.

Para melhorar a eficiência em diferentes aplicações, o RSVP oferece diversos tipos de reserva, que determinam o modo como é feita a agregação das reservas: *wildcard*, filtro fixo (*fixed-filter*) e filtro dinâmico (*dynamic-filter*). A reserva tipo *wildcard* especifica que a reserva abrange todas as fontes a emitir na sessão, permitindo uma partilha optimizada de recursos em aplicações onde apenas um conjunto restrito de emissores estão activos em cada momento, como acontece numa sessão de áudio-conferência. Em reservas de tipo filtro fixo cada um dos emissores é explicitamente seleccionado, bem como a quantidade de recursos afectos a tráfego desse emissor. Reservas deste tipo não se podem alterar sem nova submissão a controlo de admissão. Nas reservas com filtro dinâmico é explicitamente identificado um conjunto de emissores cujo tráfego vai partilhar os recursos, sendo esse conjunto alterável dinamicamente sem nova submissão a controlo de admissão desde que os recursos sejam suficientes. Informação mais detalhada sobre os tipos de reserva pode encontrar-se em [4].

O RSVP define vários tipos de mensagem: *Path*, *Resv*, *PathErr*, *ResvErr*, *PathTear*, *ResvTear* e *ResvConf*. De entre estas, as mais importantes são *Path* e *Resv*, usadas para estabelecer sessões RSVP com QoS. Cada mensagem RSVP contém um cabeçalho que identifica o tipo de mensagem e o seu comprimento, sendo o resto da mensagem constituída por *objectos*. Estão definidos os seguintes *objectos* para as mensagens RSVP:

NULL	A ignorar pelo destino
SESSION	Identificador da sessão (destino do fluxo)
RSVP_HOP	Endereço do salto[39] anterior (PHOP) ou seguinte (NHOP)
TIME_VALUES	Período de refrescamento das mensagens *Path* e *Resv*
STYLE	Tipo de reserva
FLOWSPEC	QoS desejada
FILTER_SPEC	Identificação dos pacotes abrangidos pela reserva
SENDER_TEMPLATE	Identificação do emissor a nível de transporte

[39] *Hop*

SENDER_TSPEC	Limite superior do tráfego emitido
ADSPEC	Informação cumulativa sobre as características de QoS ao longo do percurso
ERROR_SPEC	Informação de erro
POLICY_DATA	Informação opaca ao RSVP, para ser tratada pelo módulo de controlo administrativo
INTEGRITY	Informação de autenticação e verificação (MD5)
SCOPE	Lista de routers em direcção à fonte, usada para evitar laços de encaminhamento
RESV_CONFIRM	Endereço para onde deve ser enviada a confirmação da reserva

Cada emissor de uma sessão RSVP deve, periodicamente, enviar mensagens *Path* para o mesmo destino do tráfego do fluxo. Apesar de não fazerem parte do fluxo, estas mensagens seguem o mesmo percurso que ele, e a sua função é precisamente estabelecer esse percurso. As mensagens *Path* incluem obrigatoriamente, além do cabeçalho RSVP, os objectos SESSION, RSVP_HOP, TIME_VALUES e SENDER_TSPEC, podendo, opcionalmente, incluir os objectos INTEGRITY, POLICY_DATA, SENDER_TEMPLATE e ADSPEC.

A reserva de recursos é efectuada pelo receptor, através do envio de mensagens *Resv*. Estas vão seguir o percurso inverso das mensagens *Path*, pelo que é impossível efectuar qualquer reserva se o emissor não tiver previamente enviado nenhuma mensagem *Path*. As mensagens *Resv* incluem obrigatoriamente o cabeçalho RSVP e os objectos SESSION, RSVP_HOP, TIME_VALUES, STYLE, FLOWSPEC e FILTER_SPEC. Opcionalmente podem incluir os objectos INTEGRITY, RESV_CONFIRM, SCOPE e POLICY_DATA.

As mensagens *PathErr* e *ResvErr* indicam, respectivamente, falha em mensagens *Path* ou *Resv*. Apesar de as reservas serem voláteis, é mais eficiente terminar activamente as reservas, sendo esta a função das mensagens *PathTear* e *ResvTear*. As mensagens *ResvConf* confirmam a efectivação de uma reserva, se essa confirmação for solicitada através do objecto RESV_CONFIRM na mensagem *Resv*.

O *Adspec* é um objecto que pode, opcionalmente, ser incluído pelo emissor nas mensagens *Path*, sendo alterado nos *routers* RSVP ao longo do percurso, o que vai permitir ao receptor conhecer as características extremo-a-extremo desse mesmo percurso por forma a determinar os recursos que necessita de reservar para obter a qualidade de serviço desejada. Este objecto contém obrigatoriamente um fragmento de Parâmetros Genéricos Padrão (*Default General Parameters*) e pelo menos um dos seguintes: fragmento de Serviço Garantido e fragmento de Serviço de Carga Controlada. A omissão de qualquer destes últimos implica que o serviço correspondente não está disponível, sendo esta uma forma de o emissor forçar todos os receptores a usarem o mesmo tipo de serviço. Isto pode ser útil, uma vez que reservas respeitantes a serviços de tipo diferente não podem

ser agregadas[40]. O fragmento de Parâmetros Genéricos Padrão contém os seguintes campos:

- Latência mínima do percurso, que representa o atraso sofrido por um pacote ao atravessar o percurso na ausência de espera em filas.
- Débito do percurso, que é igual ao do troço de menor débito que integra o percurso.
- *Bit* Global de Quebra, indicando a existência de pelo menos um troço do percurso em que o RSVP não é suportado, podendo invalidar o conteúdo do objecto *Adspec*.
- Contador de saltos de serviços integrados, que é incrementado na passagem em cada *router* que suporte RSVP ao longo do percurso.
- MTU do percurso, representando o valor mínimo de todas as MTU individuais ao longo do percurso.

O fragmento de Serviço Garantido inclui os seguintes campos:

- C_{tot}, valor composto, extremo-a-extremo, do anteriormente referido parâmetro de erro C.
- D_{tot}, valor composto, extremo-a-extremo, do parâmetro de erro D.
- C_{Sum}, valor composto, desde o último ponto de re-formatação, do parâmetro C.
- D_{Sum}, valor composto, desde o último ponto de re-formatação, do parâmetro D.
- *Bit* de Quebra de Serviço Garantido, indicando a existência de um *router* no percurso que suporta RSVP mas não o Serviço Garantido.

Opcionalmente, pode ainda existir um campo com valores genéricos que se irão sobrepor aos contidos no fragmento de Parâmetros Genéricos Padrão para reservas com Serviço Garantido. O fragmento de Serviço de Carga Controlada apenas contém um *Bit* de Quebra de Serviço de Carga Controlada, com significado idêntico ao seu congénere para Serviço Garantido, podendo opcionalmente existir um campo com valores genéricos a sobrepor aos contidos no fragmento de Parâmetros Genéricos Padrão em reservas de Serviço de Carga Controlada.

A existência do *Adspec* permite que se efectuem reservas usando o método *One Pass With Advertising* (OPWA)[41], em que o receptor pode conhecer *a priori* com precisão a qualidade de serviço resultante de uma dada reserva, ou que recursos necessita de reservar para obter a qualidade de serviço desejada. Para o Serviço Garantido, por exemplo, a aplicação pode subtrair a latência mínima do percurso ao valor máximo desejado para o atraso e usar o valor obtido, bem como os valores de C_{tot} e D_{tot}, na fórmula (2.1) para calcular o valor de débito, R, que é necessário reservar.

Em determinadas condições, a reserva efectuada por um *router* e transmitida ao seguinte em direcção ao emissor pode conter um valor de débito inferior ao que lhe foi solicitado. Isto permite

[40] Reservas do mesmo tipo mas com parâmetros diferentes, no entanto, são agregadas pelo RSVP.
[41] Por oposição às *One Pass*, efectuadas na ausência deste objecto.

flexibilizar a reserva de recursos em *routers* cujo algoritmo de escalonamento trate os parâmetros de atraso e débito independentemente. Isto pode permitir aceitar determinados pedidos de QoS que de outro modo seriam rejeitados. Isto é possível desde que o atraso adicional introduzido por uma reserva inferior não exceda o valor S de folga, devendo respeitar a seguinte condição, que relaciona os parâmetros de saída com os de entrada:

$$S_o + \frac{b}{R_o} + \frac{C_{toti}}{R_o} \leq S_i + \frac{b}{R_i} + \frac{C_{toti}}{R_i}$$

em que $r \leq R_o \leq R_i$. C_{toti} é o valor combinado do parâmetro C no conjunto de *routers* em direcção ao emissor, incluindo ele próprio. Note-se que o *router* conhece o valor de r, uma vez que este faz parte de *TSpec*.

2.5.5 Prós e contras do modelo de Serviços Integrados

O modelo de serviços integrados permite obter qualidade de serviço por fluxo, através de reserva. As garantias que oferece podem ser rígidas (Serviço Garantido), ou meramente estatísticas (Serviço de Carga Controlada), consoante as necessidades da aplicação.

Apesar de o modelo de Serviços Integrados ter sido concebido como uma extensão ao modelo arquitectónico em que se baseia a Internet, o facto de ser necessário suportá-lo em todos os *routers* ao longo do percurso do tráfego para as garantias poderem ser efectivas é um factor determinante, pois é improvável que venha a ser implementado em larga escala. Isto porque sendo as garantias atribuídas por fluxo, obriga os *routers* a manter e utilizar informação de estado, que cresce na proporção directa do número de fluxos, o que dificulta ou mesmo impossibilita o seu uso nos *backbones*, dado o elevado número de fluxos que transportam em simultâneo. Foi este problema de escalabilidade que tornou necessária a concepção de soluções alternativas para a qualidade de serviço na Internet, nomeadamente o modelo de Serviços Diferenciados.

2.6 Serviços Diferenciados

O modelo de Serviços Diferenciados, abreviadamente *diffserv*, surgiu como uma aproximação mais escalável à qualidade de serviço, evitando a manutenção de informação de estado por microfluxo nos nós interiores da rede, onde o seu número é potencialmente muito elevado. Embora funcione ao nível 3 como o *intserv*, no modelo *diffserv* fluxos com os mesmos requisitos são agregados no interior da rede, e a informação de estado dos agregados de tráfego é transportada no próprio cabeçalho IP, no campo DS, sendo usada nos nós de encaminhamento como forma simples e eficiente de classificar os pacotes para diferenciar o tratamento por eles recebido.

A diferenciação de serviços não é em si uma ideia nova. De facto, muito antes do aparecimento do modelo *diffserv* existiram diversas aproximações à diferenciação de serviços, nomeadamente a nível do IP, com a indicação explícita de uma prioridade relativa ou a indicação do tipo de serviço

desejado[42] que influenciaria a escolha do percurso, campos incluídos no octeto TOS do cabeçalho IP. Estas aproximações nunca tiveram, no entanto, uma implementação em larga escala, em parte devido a algumas limitações que o modelo de serviços diferenciados visa colmatar. De facto, o modelo *diffserv* é muito mais flexivel que um mecanismo de simples prioridade, uma vez que a diferenciação pode ser feita a nível de escalonamento, gestão de filas, selecção de rota ou qualquer outro factor que possa influenciar o desempenho do serviço. Por outro lado, disponibiliza um número de serviços possíveis muito superior à diferenciação baseada no tipo de serviço. Além disso, o mapeamento entre a marcação do pacote e o tipo de serviço a prestar-lhe não é estático, podendo adaptar-se às necessidades de cada rede em particular.

2.6.1 Modelo arquitectónico

Um dos conceitos chave do modelo arquitectónico *diffserv* é o de comportamentos por salto (*per-hop behaviours* — PHB), que constituem as peças base usadas na implementação de serviços. O comportamento por salto é uma descrição do tratamento, conforme observado externamente, que é dado aos pacotes pertencentes a um dado agregado, isto é, que contêm o mesmo valor no campo DS. Na maior parte dos casos, esta descrição apenas tem significado quando existe simultaneamente tráfego de outros agregados, assumindo que o algoritmo de escalonamento não deixa a ligação desocupada enquanto as filas de espera não estiverem vazias. É a partir desta descrição que são reservados recursos para os agregados de tráfego nos nós da rede.

Um exemplo extremamente simples de comportamento por salto é a garantia de uma percentagem mínima do débito da ligação para um dado agregado de tráfego. Um outro exemplo um pouco mais complexo é o de garantir uma percentagem mínima do débito e uma distribuição proporcional da capacidade excedente, quando esta existir. Em geral, o comportamento observável de um PHB pode depender não só das características de tráfego desse agregado, mas também dos outros agregados com os quais a ligação é partilhada.

Nos casos em que existe uma interdependência na especificação de comportamentos por salto, a sua especificação pode ser agrupada, constituindo o que se designa um grupo de PHB. A diferenciação entre os vários PHB do grupo pode ser feita com base em prioridades, absolutas ou relativas, de acesso aos recursos, ou das características observáveis do tráfego, como o atraso introduzido ou a probabilidade de perdas.

Nos nós da rede, os PHB são implementados com base essencialmente em mecanismos de gestão de filas e escalonamento de pacotes. No entanto, a especificação de PHB é feita com base nas características comportamentais relevantes para o serviço e não nos mecanismos usados para a implementar, podendo, em geral, ser usados diferentes mecanismos para implementar um mesmo PHB.

Num mesmo nó da rede podem ser suportados diversos grupos de PHB. Assim, deve ser possível inferir a quantidade de recursos a reservar para os diversos grupos de PHB suportados, bem como a forma de integrá-los, a partir da sua definição. Se houver qualquer incompatibilidade entre um

[42] Atraso mínimo, débito máximo, fiabilidade máxima ou custo mínimo.

grupo de PHB e outro anteriormente especificado, essa incompatibilidade deve ser explicitamente indicada.

Para associar cada pacote a um dos PHB implementados, os nós têm que fazer a classificação dos pacotes que lhes são entregues. Nos nós interiores da rede é importante que a classificação seja feita de forma simples e eficiente, sem necessidade de analisar múltiplos campos dos cabeçalhos, possivelmente de camadas diferentes. O modelo *diffserv* redefiniu o octeto TOS do cabeçalho IPv4 (e o octeto de Classe de Tráfego, no caso do IPv6) como octeto DS. Este octeto contém dois campos: o DSCP (*Differentiated Services CodePoint*), com 6 *bit*, usado na determinação do PHB, e dois *bit* CU (*Currently Unused*), reservados para utilização futura, e que devem ser ignorados para efeitos de selecção de PHB.

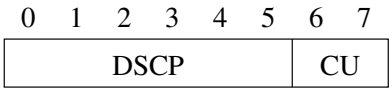

Figura 2.4: Estrutura do octeto DS do cabeçalho IP.

O campo DS está definido de forma não estruturada para facilitar a introdução de novos PHB no futuro, evitando as limitações inerentes ao antigo campo TOS. Assim, uma implementação do modelo *diffserv* deve usar o campo DSCP na sua totalidade para seleccionar o PHB. Existe, no entanto, uma divisão no espaço dos DSCP possíveis, reservando uma parte para normalização de PHB (DSCP = xxxxx0)[43] e outra para uso experimental ou uso local pela rede (DSCP = xxxx11). Uma outra parte desse espaço (DSCP = xxxx10) está presentemente reservada para uso local/experimental, estando, no entanto, salvaguardada a possibilidade de se vir a usar para PHB normalizados no caso de se esgotar o espaço para eles reservado.

O modelo *diffserv* obriga à existência de um PHB padrão, correspondente ao tratamento tipo melhor esforço conforme descrito em [7]. Assim, as aplicações que não suportem *diffserv* podem continuar a ser usadas sem uma significativa degradação do serviço. O valor do DSCP recomendado para este PHB é 000000. O valor do DSCP para este PHB pode ser diferente, mas pacotes com DSCP = 000000 têm que ser sempre mapeados para um PHB com os mesmos requisitos que o PHB padrão.

Embora o modelo *diffserv* não pretenda manter qualquer compatibilidade com a utilização do campo TOS definida em [8], é mantida uma certa compatibilidade com o campo de prioridade conforme definido em [9], que se sabe ter hoje alguma implantação. Para manter essa compatibilidade foi definido um conjunto de PHB, ao qual estão atribuídos os designados Códigos Selectores de Classe (Class Selector Codepoints), com o campo DSCP = xxx000. Os requisitos para estes PHB são:

1. A probabilidade de entrega atempada de pacotes com um dado Código Selector de Classe não deve ser, em condições razoáveis de operação da rede, inferior ao de pacotes com um outro

[43] Usando a notação "xxxxxx", onde cada "x" corresponde a um *bit* do DSCP.

Código Selector de Classe numericamente inferior[44].

2. Pacotes com marcação DSCP = 11x000 devem obrigatoriamente ter tratamento preferencial em relação a pacotes com DSCP = 000000[45].

Adicionalmente, deve ter-se em conta que o tratamento de pacotes com diferentes Códigos Selectores de Classe deve ser feito de forma independente, pelo que a sua ordem temporal pode ser alterada.

Dois outros conceitos importantes na arquitectura *diffserv* são os de SLA (*Service Level Agreement*) e TCA (*Traffic Conditioning Agreement*). O SLA é um contrato entre um cliente e um fornecedor de serviço contendo uma especificação do serviço a ser prestado, podendo o cliente ser uma organização com utilizadores finais ou um outro fornecedor de serviço. Um SLA pode incluir regras de condicionamento de tráfego constituindo um TCA. Um TCA é um contrato onde são especificadas regras de classificação, perfis de tráfego e regras para medição, marcação, eliminação e/ou formatação de pacotes.

Na arquitectura *diffserv* é feita uma clara distinção entre nós fronteira e nós interiores de um domínio DS. Um domínio DS é um conjunto de nós contíguos com suporte a DS, aos quais é comum uma política de fornecimento de serviços e o conjunto de PHB implementados, sendo normalmente constituído por uma ou mais redes sob a mesma administração. Os nós fronteira, responsáveis pela interligação com outros domínios, que podem suportar ou não *diffserv*, funcionam quer como nós de ingresso, quer como nós de egresso, para tráfego em sentidos inversos. Um nó de ingresso é responsavel por garantir que o tráfego que entra na rede está conforme ao TCA existente entre o domínio DS e o domínio exterior (suportando ou não DS). Um nó de egresso é responsável por eventuais funções de condicionamento de tráfego existentes entre os dois domínios contíguos. Os nós fronteira são responsáveis pela marcação/remarcação dos pacotes. Nestes nós podem ser usados os designados classificadores multi-campo (*multi-field* — MF), que se baseiam em múltiplos campos do cabeçalho (endereços de origem e destino, protocolos, portas, etc.) para atribuír os pacotes a agregados de tráfego com PHB suportados nesse domínio, marcando-os com o DSCP apropriado. Os nós internos, por razões de eficiência, apenas usam o DSCP para classificação (classificadores BA — *behaviour aggregate*), trabalhando apenas com agregados.

Para garantir conformidade do tráfego com o(s) TCA aplicáveis, os nós fronteira possuem condicionadores de tráfego, que podem incluir os seguintes elementos:

Medidor (*meter*): Os medidores determinam as características temporais de fluxos (agregados). Após comparação dessas características com o perfil de tráfego determinado pelo TCA, é passada infomação de estado àos restantes elementos para que tomem as acções apropriadas, conforme o pacote corresponda ou não ao perfil[46].

[44] Para todos os efeitos, a perda de um pacote pode considerar-se um caso extremo de não entrega atempada.

[45] Isto para manter compatibilidade com o uso comum dos valores de precedência IP 110 e 111 para tráfego dos protocolos de encaminhamento.

[46] Note-se que é possível haver mais níveis de conformidade do que simplesmente "conforme" e "não-conforme".

Marcador (*marker*): A função dos marcadores é colocar um valor no campo DSCP dos pacotes, atribuíndo-os ao agregado correspondente, determinando assim o PHB a aplicar-lhes.

Formatador (*shaper*): Os formatadores atrasam alguns pacotes (ou mesmo todos) de um fluxo (ou agregado) para forçar a sua conformidade com o perfil adequado. Em princípio, isto é feito com recurso a um *buffer*, que por ter um espaço finito pode levar a perdas se este se esgotar.

Eliminador (*dropper*): Este elemento destina-se a eliminar alguns pacotes de fluxos (agregados) por forma a torná-los conformes ao perfil, ou seja, faz policiamento. Isto é necessário para evitar a ocorrência de congestionamento. O eliminador pode ser implementado como um caso especial de formatador em que o *buffer* é extremamente pequeno ou inexistente.

Num domínio DS em que exista equipamento terminal, ele pode actuar como nó fronteira para o seu tráfego. Se o equipamento não suportar essa funcionalidade, deve ser o *router* mais próximo a fazê-lo.

Um conjunto contíguo de domínios DS com SLA estabelecidos entre si que definam os TCA necessários ao suporte de um conjunto comum de grupos de PHB designa-se região DS. As regiões DS suportam um conjunto de serviços ao longo de qualquer rota que as atravesse. Internamente, cada um dos domínios pode suportar diferentes mapeamentos DSCP→PHB. Os nós fronteira devem ser, portanto, capazes de fazer a remarcação para compensar estas diferenças.

Feita uma descrição do modelo arquitectónico diffserv, serão apresentados de seguida dois grupos de PHB actualmente normalizados: *Assured Forwarding* [10] e *Expedited Forwarding* [11].

2.6.2 *Assured Forwarding*

O grupo AF (*Assured Forwarding*) de PHB visa o fornecimento de classes distintas de tráfego com níveis variáveis de probabilidade de perda. Para tal, preconiza a existência de quatro classes AF distintas, cada uma com recursos próprios atribuídos, e encaminhadas independentemente. Dentro de cada uma das classes, existem três níveis de precedência de perda, correspondendo a uma maior ou menor probabilidade de perda dentro da classe. Este grupo de PHB garante que pacotes de um mesmo microfluxo que estejam atribuídos à mesma classe AF não são reordenados, ainda que tenham níveis de precedência de perda diferentes.

Designa-se por AFij o agregado AF da classe i e com precedência de perda j. A tabela 2.2 apresenta os DSCP recomendados para cada um dos agregados AFij.

Em cada nó devem obrigatoriamente existir certos níveis mínimos de recursos afectos a cada classe, por forma a disponibilizar-lhe o serviço contratado em intervalos de tempo grandes e pequenos. Dentro de cada classe, um nó não pode eliminar com maior probabilidade de perda pacotes com precedência de perda menor. É possível um nó implementar apenas dois níveis de perdas, mas neste caso os pacotes AFx1 devem ser mapeados no nível de perdas mais baixo e os AFx2 e AFx3 no nível de perdas mais elevado.

	Classe 1	Classe 2	Classe 3	Classe 4
Baixa precedência de perda (1)	001010	010010	011010	100010
Média precedência de perda (2)	001100	010100	011100	100100
Elevada precedência de perda (3)	001110	010110	011110	100110

Tabela 2.2: DSCP recomendados para o grupo AF

Em nós que suportem AF, períodos curtos de congestionamento devem ser absorvidos pela utilização de *buffers*. Em escalas temporais mais largas, o congestionamento deve ser controlado através da eliminação de pacotes. Esta deve ser, no entanto, gradual e não abrupta, o que implica que os pacotes não devem ser eliminados apenas quando se esgota o espaço em *buffer*, devendo implementar um mecanismo semelhante ao RED. Por outro lado, o algoritmo de eliminação de pacotes deve ser configurável por precedência de perda e por classe, pelo que o GRED é o mais adequado. Os algoritmos RED e GRED são abordados na secção 3.8.

2.6.3 *Expedited Forwarding*

O PHB EF (*expedited forwarding*) destina-se a suportar serviços como o de linha dedicada virtual. Um serviço deste tipo caracteriza-se por baixos valores de perdas, latência e *jitter*. Isto implica que em cada um dos nós os pacotes deste agregado devem encontrar as filas de espera vazias ou com uma ocupação muito baixa, implicando uma taxa de serviço superior à taxa de chegada. Um tal serviço implica duas coisas:

1. Em cada nó a taxa de serviço deve ser fixa e independente da intensidade de outro tráfego que o atravesse.

2. O tráfego deve ser formatado e/ou policiado por forma a que a taxa de chegada não exceda em nenhum nó a taxa de serviço.

Enquanto as funções de condicionamento de tráfego dos nós fronteira se encarregam da segunda condição, a primeira é assegurada pelo PHB EF.

Um nó que implemente o PHB EF deve garantir-lhe o débito configurado em intervalos de tempo iguais ou superiores ao que levaria a transmissão de um pacote de tamanho máximo com esse débito. Em termos de implementação, o suporte ao PHB EF implica capacidade de preempção relativamente a outros agregados de tráfego, através, por exemplo, de um mecanismo de prioridades. No entanto, para que o tráfego deste agregado não influencie negativamente o de outros, deve existir um limite superior para o seu débito. Pacotes que excedam este limite deverão obrigatoriamente ser eliminados.

Pacotes marcados como EF podem ser remarcados nos nós fronteira, mas apenas para valores de DSCP a que corresponda um PHB EF, não devendo ser atribuídos a agregados com PHB diferente. O valor do DSCP recomendado para o PHB EF é o 101110.

Capítulo 3

Serviços diferenciados: controlo de tráfego

Na implementação do modelo de Serviços Diferenciados, bem como de qualquer arquitectura de rede com qualidade de serviço, é necessário um módulo de controlo de tráfego, fazendo uso de algoritmos de escalonamento e outros mecanismos que permitam dar aos pacotes um tratamento diferenciado. Neste capítulo são apresentados diversos algoritmos de escalonamento de pacotes e mecanismos de gestão de filas de espera usados na implementação dos PHB que permitem a configuração de diferentes serviços no modelo *diffserv*.

3.1 FIFO

O FIFO, *First In, First Out*, é o algoritmo de escalonamento mais básico, tendo sido usado desde sempre, particularmente em redes que disponibilizam um serviço tipo melhor esforço. Este algoritmo baseia-se no armazenamento de pacotes numa fila de espera, efectuando a sua transmissão por ordem de chegada quando há disponibilidade da ligação. Pretende-se com isto evitar a perda de pacotes em períodos curtos de congestionamento, absorvidos pela utilização da fila de espera.

O algoritmo FIFO não prevê qualquer diferenciação no tratamento dado aos pacotes: é a ordem de chegada que determina o débito do fluxo, o espaço utilizado em *buffer* e o atraso sofrido pelo pacote no nó. Assim, não há qualquer garantia de transmissão do pacote ao nó seguinte ou limite ao atraso introduzido, uma vez que não há qualquer protecção contra fluxos "mal comportados". Isto não significa que o FIFO seja inútil na implementação de serviços diferenciados, uma vez que pode ser usado no tratamento de pacotes pertencentes a um mesmo fluxo ou agregado de fluxos.

Em termos de complexidade e consumo de recursos, nomeadamente de processamento, o FIFO é o algoritmo de escalonamento mais simples e eficiente.

3.2 PRIO

O escalonamento por prioridade, abreviadamente PRIO, foi concebido para permitir que o tráfego de maior importância seja transmitido em primeiro lugar. O escalonamento por prioridade baseia-se na existência não de uma, mas de várias filas de espera, estando atribuído a cada fila um valor de prioridade. Com base numa classificação feita no nó, é determinada a prioridade do pacote, sendo este colocado na fila de espera correspondente. Internamente, cada uma das filas tem um tratamento FIFO, mas só são transmitidos pacotes de uma determinada fila quando todas as de prioridade superior estiverem vazias.

Embora o escalonamento por prioridade garanta que pacotes com maior importância têm uma probabilidade de perda inferior e sofrem um menor atraso que os de importância inferior, não há qualquer garantia que entre fluxos de tráfego com igual prioridade não exista o mesmo tipo de problemas de um serviço tipo melhor esforço. Além disso, um algoritmo de prioridades, quando usado isoladamente, não impede o tráfego de maior prioridade de ocupar a totalidade da capacidade disponível, impedindo o de prioridade inferior de ser transmitido.

O escalonamento por prioridade tem, no entanto, a grande vantagem de consumir poucos recursos. De facto, a única tarefa que pode ter alguma complexidade é a classificação dos pacotes, o que torna o PRIO num dos algoritmos mais simples e eficientes na implementação de mecanismos de qualidade de serviço.

3.3 WRR e DWRR

O algoritmo WRR (*weighted round-robin*) pretende solucionar o problema da partilha de uma mesma ligação por diversos fluxos (ou agregados de fluxos), evitando que um deles monopolize a sua utilização em prejuízo dos outros. Tal como no caso anterior, o WRR baseia-se na existência de várias filas de espera, sendo os pacotes colocados numa delas após classificação. A cada uma dessas filas de espera é atribuído um determinado peso. Para transmissão, as várias filas de espera são percorridas de forma circular, transmitindo de cada fila um volume de tráfego proporcional ao peso que lhe foi atribuído. Se uma dada fila estiver vazia, passa a processar-se a seguinte para garantir que enquanto houver tráfego para transmitir a ligação não fica desocupada.

Este algoritmo, na sua versão mais básica, tem um problema: sempre que o tamanho de um pacote excede a quantidade de dados dessa fila que ainda pods ser transmitida nessa passagem, esse pacote tem que esperar pela passagem seguinte. Assim, essa capacidade diferencial foi perdida pelo fluxo pelo simples facto de ter no princípio da fila um pacote maior. Em certas condições este efeito pode tornar-se cumulativo, fazendo com que o débito efectivo desse fluxo (ou agregado de fluxos) seja inferior àquele que seria expectável dado o seu peso. Existe, no entanto, uma variante, designada DWRR (*deficit weighted round-robin*) que contorna este problema guardando informação sobre esse diferencial de capacidade, que é adicionado à quantidade de dados dessa fila a transmitir na passagem seguinte. Com isto consegue-se uma distribuição do débito que em média[1] é sempre

[1] Em janelas temporais da ordem de grandeza de uma passagem por todas as filas, no entanto, pode não ser.

correspondente à proporção dos pesos, à custa de um ligeiro aumento do atraso máximo[2].

Como qualquer algoritmo da família *round-robin*, o WRR (ou DWRR) é extremamente eficiente em termos de processamento, sendo os seus requisitos em termos de memória e CPU mínimos, pelo que a sua utilização na implementação de serviços não põe quaisquer problemas de escalabilidade.

3.4 WFQ, WF^{2}Q, WF^{2}Q+

O algoritmo WFQ (*weighted fair queuing*) é uma aproximação ao modelo idealizado GPS[3] (*generalized processor sharing*), baseado num modelo fluido de tráfego e, por isso, não aplicável directamente a redes de comutação de pacotes. O WFQ baseia-se no uso de uma função de tempo virtual. Quando o pacote k do fluxo (ou agregado) i chega ao nó, são calculados os instantes virtuais de início e fim da sua transmissão, $S(i,k)$ e $F(i,k)$, correspondentes aos instantes que se verificariam no modelo GPS.

$$\begin{aligned} S(k,i) &= \max\left(F\left(k-1,i\right), V\left(a\left(k,i\right)\right)\right) \\ F(k,i) &= S(k,i) + \frac{L(k,i)}{r(i)} \end{aligned}$$

$a(k,i)$ é o instante de chegada ao nó do pacote (k,i); $L(k,i)$ é o seu comprimento; $r(i)$ é o débito reservado para o fluxo (ou agregado) i. $V(t)$ é uma função de tempo virtual que simula o tempo no modelo GPS emulado, e cuja derivada temporal é igual ao inverso do somatório das fracções de débito atribuídas aos fluxos activos no instante t. Isto implica que quando um fluxo (ou agregado) está inactivo, a função de tempo virtual cresce mais rapidamente, pelo que os restantes fluxos recebem um melhor serviço, na proporção directa da fracção de débito que lhes está atribuída. Sempre que é possível transmitir um pacote, o sistema escolhe o que tiver o menor valor $F(k,i)$.

Em [12] é apresentado o algoritmo PGPS, idêntico ao WFQ, e são demonstradas as seguintes propriedades:

1. O fim de transmissão de um pacote no sistema WFQ não se atrasa mais que o tempo de transmissão de um pacote de tamanho máximo em relação ao sistema GPS.

2. O número de *bits* de um fluxo transmitidos no WFQ não é inferior ao número de *bits* do mesmo fluxo transmitidos no GPS menos o tamanho de um pacote de tamanho máximo.

Assim, o serviço de pacotes em WFQ não é significativamente inferior ao que se verifica no modelo idealizado GPS, pelo que é possível utilizá-lo na implementação de serviços garantidos. Em particular, o WFQ é utilizado na implementação do modelo de Serviços Integrados.

Apesar de o serviço em WFQ não poder ser pior que em GPS a menos de um pacote de tamanho máximo, em determinadas condições pode dar a determinados pacotes um serviço bastante melhor em termos de atraso. Além de minar o pressuposto de equidade, isto pode aumentar a *burstiness*,

[2]No pior caso, o atraso introduzido pelo DWRR pode aproximar-se do dobro do introduzido pelo WRR, mas na práctica a diferença é inferior.

[3]De facto, o WFQ é idêntico ao PGPS, versão do GPS para redes de comutação de pacotes.

o que é indesejável. O algoritmo WF^{2}Q (*worst-case fair weighted fair queuing*) é uma melhor aproximação ao GPS neste ponto. A diferença entre o WFQ e o WF^{2}Q é que neste último, quando é seleccionado um pacote para transmissão, apenas são levados em conta os pacotes cujo instante virtual de início de transmissão não seja posterior ao valor da função de tempo virtual no instante presente. Assim melhora-se o desempenho global do sistema, mas à custa de um ligeiro aumento de complexidade de processamento, que já é bastante elevada no caso do WFQ.

Para reduzir a complexidade do WF^{2}Q mantendo as suas características interessantes, foi apresentado em [13] um outro algoritmo, designado WF^{2}Q+, que apresenta uma complexidade O(log(N)) em vez de O(N) como acontece nos casos anteriores. Este algoritmo utiliza uma nova função de tempo virtual

$$V(t+\tau) = \max(V(t) + W(t, t+\tau), \min_{i \in \widehat{B}(t)} (S(h(t,i),i))$$

em que $W(t+\tau)$ é o serviço total prestado pelo nó no intervalo de tempo $[t, t+\tau]$, $\widehat{B}(t)$ é o conjunto de fluxos (ou agregados) com pacotes pendentes no instante t, $h(t,i)$ é o número de sequência do primeiro pacote pendente do fluxo i, e $S(h(t,i),i)$ é o seu instante virtual de início de transmissão. São ainda demonstradas em [13] as propriedades deste algoritmo.

Embora as características de atraso sejam melhores nestes algoritmos que no (D)WRR[4], a sua maior complexidade[5] é um factor limitativo à sua implementação em nós que suportem elevados volumes de tráfego.

3.5 CSZ[6]

Em [14] é proposto um algoritmo de escalonamento de pacotes que permite suportar simultaneamente e de forma eficiente a integração de serviços garantidos, destinados ao transporte de tráfego com requisitos de tempo real, serviços previsíveis, para serviços sensíveis ao atraso mas adaptativos, e um serviço tipo melhor esforço, para o restante tráfego, numa mesma rede de comutação de pacotes. Este algoritmo baseia-se na existência de N filas de espera, uma por cada fluxo (ou agregado) com serviço garantido. A cada uma destas filas é atribuída uma certa fracção do débito disponível, sendo o restante atribuído a um pseudo-fluxo 0 que engloba todo o tráfego de serviço previsível e de tipo melhor esforço. Estes $N+1$ fluxos são servidos segundo a disciplina WFQ anteriormente descrita. Para o pseudo-fluxo 0 não existe uma fila de espera única, mas sim um conjunto de filas de espera servidas entre si segundo uma disciplina de prioridade estrita. Isto permite o fornecimento de serviços previsíveis com diferentes níveis de *jitter*, uma vez que com este tipo de serviço o *jitter* é transferido para as classes de prioridade inferior. A classe de mais baixa prioridade, que absorve

[4]No WRR, no pior caso, em que um pacote chega no instante em que o escalonador passou à fila seguinte, há um atraso igual ao somatório do tempo de transmissão das parcelas de tráfego de todas as filas. No DWRR, devido ao *deficit*, é ainda ligeiramente pior.

[5]A complexidade do DWRR é O(1).

[6]De Clark, Shenker e Zhang, proponentes do algoritmo.

o *jitter* de todas as outras, destina-se ao transporte de tráfego com serviço tipo melhor esforço. Dentro de cada classe o tráfego é servido segundo a disciplina FIFO+.

A disciplina de serviço FIFO+ baseia-se no seguinte algoritmo: quando o pacote entra num nó, é-lhe introduzida uma marca temporal; à saída é calculado, com base nessa marca temporal, o atraso sofrido nesse nó; a diferença entre este valor e o atraso médio sofrido por pacotes da mesma classe nesse nó é adicionada a um campo do cabeçalho designado para o efeito. Assim é possível em cada nó saber a diferença entre o instante em que ele chegou a esse nó e o instante em que deveria ter chegado se recebesse um serviço médio para a sua classe. O pacote é então escalonado para transmissão com base nesse instante esperado. Isto implica que, ao contrário do que acontece na disciplina FIFO, o *jitter* não aumenta dramaticamente com o número de nós no percurso, uma vez que há uma melhor distribuição do atraso que permite absorvê-lo.

O algoritmo FIFO+ apresenta, no entanto, dois problemas: por um lado, o escalonamento não é trivial como acontece no FIFO, pois a manutenção de uma fila ordenada é uma operação computacionalmente pesada; por outro lado implica a existência no cabeçalho de um campo destinado a transportar o diferencial de atraso, o que não acontece, por exemplo, no protocolo IP. Assim, uma implementação alternativa pode usar simplesmente a disciplina FIFO, abdicando dos benefícios em termos de *jitter* do FIFO+.

É de salientar que a disciplina de serviço CSZ não faz policiamento ou formatação de tráfego dos fluxos, baseando-se no pressuposto de que é feito controlo de admissão e policiamento/formatação à entrada na rede.

3.6 CBQ

O CBQ (*class-based queuing*) é um dos algoritmos mais populares na implementação de serviços diferenciados, provavelmente por não ser excessivamente complexo do ponto de vista computacional e por ser o mais flexível dos algoritmos hierárquicos de QoS. Este algoritmo visa solucionar simultaneamente os requisitos de serviços de tempo real e de partilha, de forma hierárquica, de uma ligação. A disciplina CBQ baseia-se na existência de dois escalonadores distintos: um escalonador genérico e um escalonador de partilha de ligação. O objectivo do escalonador genérico é garantir aos fluxos com requisitos de tempo real baixos valores de atraso nas filas de espera. Os objectivos do escalonador de partilha são, por um lado, impedir que as classes de tráfego de tempo real monopolizem a utilização da ligação, negando por longos períodos de tempo o acesso a ela por parte do restante tráfego e, por outro lado, fazer a distribuição do débito excedente, caso este exista, de acordo com certas regras, e não de forma arbitrária.

O CBQ baseia-se numa estrutura hierárquica em árvore das classes de tráfego. À entrada num nó que implemente o CBQ, cada pacote é classificado por forma a associá-lo a uma das folhas da estrutura hierárquica, sendo estas as únicas classes destinadas ao transporte de tráfego. As classes interiores, por sua vez, definem a forma como deve ser feita a distribuição do excedente de débito. Todas as folhas da estrutura hierárquica pertencem ao nível 1. O nível de uma classe interior é

superior em uma unidade ao do seu descendente de nível mais elevado. A figura 3.1 mostra uma possível estrutura CBQ. No canto superior esquerdo de cada classe é indicado o seu nível; no canto superior direito a percentagem de débito que lhe está afecta; no canto inferior esquerdo das classes folha é indicada a prioridade.

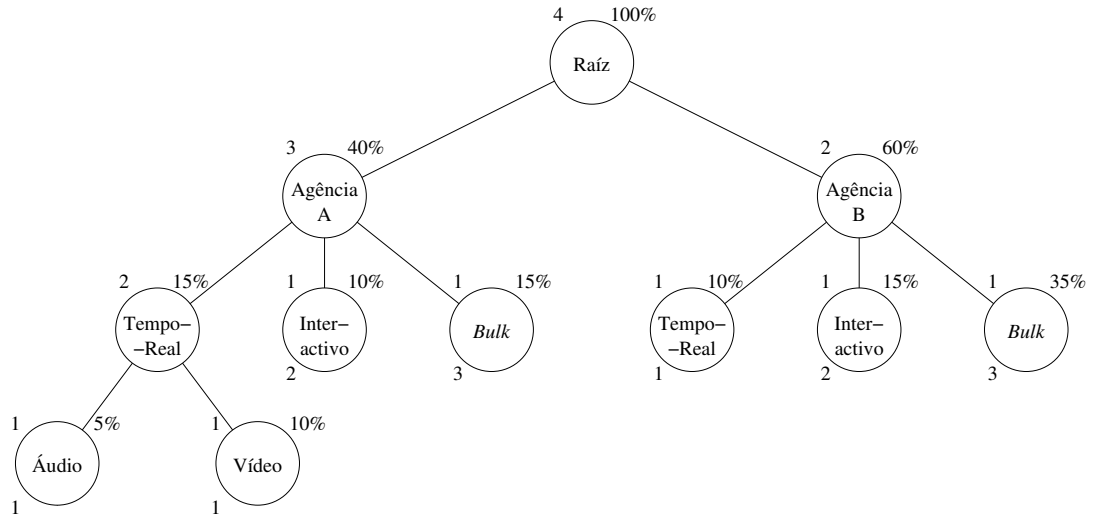

Figura 3.1: Exemplo de estrutura hierárquica CBQ

Em intervalos de tempo razoáveis, todas as classes que tenham tráfego suficente devem receber a fracção de débito que lhes foi atribuída. Existe um estimador que verifica, para cada classe, se está a transmitir abaixo, ao nível, ou acima do débito que lhe foi atribuído. Este baseia-se numa variável, *idle*, que representa a diferença entre o instante efectivo de início de transmissão de cada pacote e o instante em que isso deveria acontecer tendo em conta o débito nominal da classe. A partir deste valor é calculada uma média móvel para a classe, *avgidle*, de preferência recorrendo a uma função EWMA. Quando *avgidle* é positiva, significa que a classe está a transmitir acima do débito nominal. Com base neste resultado é feita a decisão de qual dos algoritmos vai controlar o escalonamento nessa classe. Assim, é usado o escalonador genérico sempre que:

1. A classe não está a transmitir acima do débito nominal

   ou

2. A classe tem um antepassado ao nível i que não está a transmitir acima do débito nominal e não existem classes de nível inferior a i que estejam insatisfeitas[7].

Quando nenhuma destas condições é verificada, a classe é regulada pelo escalonador de partilha. São estas as regras formais que definem o CBQ. Repare-se que, não existindo nenhuma classe insatisfeita, nenhuma outra classe é regulada pelo escalonador de partilha, mesmo que esteja a transferir acima do seu débito nominal, o que permite uma utilização eficiente da ligação.

[7]Por classe insatisfeita entende-se uma classe que está a transmitir abaixo da capacidade que lhe foi atribuída mas teria tráfego suficiente para transmitir mais, isto é, tem persistentemente pacotes na fila de espera.

Na prática, estas regras podem ser demasiado pesadas para uma implementação eficiente do CBQ. Assim, existem dois conjuntos alternativos de regras, que embora menos robustos que o conjunto de regras formais são computacionalmente mais simples. Uma das variantes diz que é usado o escalonador genérico sempre que:

1. A classe não está a transmitir acima do débito nominal

   ou

2. A classe tem um antepassado que está a transmitir abaixo do débito nominal

sendo usado o escalonador de partilha nos restantes casos. Esta variante designa-se *ancestors-only*, pois as regras apenas dependem do estado da classe e dos seus ancestrais. A outra variante, designada *top-level*, prevê o uso do escalonador genérico se:

1. A classe não está a transmitir acima do débito nominal

   ou

2. A classe tem um antepassado que está a transmitir abaixo do débito nominal, e cujo nível é, no máximo, *top-level*.

sendo usadas regras heurísticas para controlar o valor da variável *top-level*. Note-se que *top-level* tiver valor infinito esta variante é idêntica à anterior. Se, por sua vez, *top-level* contiver o nível mais baixo que contém uma classe insatisfeita, é practicamente idêntica ao CBQ com regras formais.

Para o escalonador genérico é proposto um algoritmo de prioridades, com a prioridade mais elevada atribuída aos fluxos com requisitos mais apertados de tempo real. Entre classes com a mesma prioridade é utilizado o WRR, sendo atribuído um peso proporcional à fracção do débito afecta à classe. Isto implica que quando há um excedente de débito, este é distribuído entre classes da mesma prioridade de forma proporcional ao seu débito nominal. Com este algoritmo, se às classes de prioridade 1 estiver atribuído, no máximo, até metade do débito total disponível, cada uma dessas classes recebe, garantidamente, a sua fracção de débito em cada passagem do WRR [15].

O escalonador de partilha baseia-se no princípio que se um pacote de tamanho s pertencente a uma classe à qual está atribuído o débito b for transmitido num dado instante, é necessário esperar um intervalo de tempo igual a $\frac{s}{b}$ antes de transmitir o pacote seguinte, por forma a não exceder o débito b. Na prática, este intervalo de tempo, designado *offtime*, é pré-calculado para um tamanho médio de pacote esperado, verificando-se [16] que o débito da classe, em octetos por segundo, não é significativamente afectado pelo facto de o tamanho real dos pacotes diferir do valor esperado.

Se uma classe estivesse um espaço de tempo significativo sem transmitir, a variável *avgidle* a ela associada aumentaria significativamente, e quando voltasse a transmitir teria um crédito que lhe permitiria tranferir um número potencialmente elevado de pacotes em rajada. Para evitar os problemas que isto traria às outras classes, existe um parâmetro *maxidle* que, limitando o crédito

máximo a uma classe que esteve sem transmitir, limita o tamanho máximo da rajada de pacotes que pode enviar. Por outro lado, uma classe (folha) que usou algum débito excedente não deve ser penalizada por isso, pelo que quando uma folha da hierarquia tem um valor *avgidle* negativo, este é reposto a zero. Para que as classes interiores não sejam demasiado penalizadas por usarem débito excedente, existe ainda um parâmetro, *minidle*, que limita inferiormente o valor de *avgidle*, limitando assim essa penalização.

O CBQ não visa, por si só, resolver todos os problemas de congestionamento, uma vez que dentro de cada classe esse problema é deixado em aberto. No entanto este problema pode aliviar-se utilizando dentro de cada classe um algoritmo como o RED, descrito na secção 3.8.

Em relação ao CSZ, o CBQ apresenta-se como uma alternativa mais flexível e eficiente, embora menos precisa no tratamento de fluxos com serviço garantido.

3.7 Fair Queuing e SFQ

Tanto o algoritmo original de *fair queuing* (FQ) como a versão optimizada SFQ (*stochastic fairness queuing*) visam não a diferenciação de serviço entre pacotes, mas uma distribuição equitativa do débito disponível pelos fluxos que partilham a ligação. Um método simples de atingir este objectivo seria ter uma fila de espera por cada fluxo, transmitindo um pacote de cada fila de forma circular. No entanto, tamanhos diferentes de pacotes implicariam uma distribuição injusta do débito disponível, além de que o número de filas de espera tem que ser igual ao número total de fluxos possíveis. Para evitar esses problemas, o algoritmo FQ aproxima o funcionamento de um BR (*bit-by-bit round-robin*), embora baseado em pacotes. Quando um pacote chega ao nó é calculado o instante em que terminaria a sua transmissão se o algoritmo usado fosse o BR, sendo inserido na única fila de espera de modo a que esta se mantenha ordenada em relação aos instantes calculados. Sendo uma emulação do BR, o débito é distribuído de forma equitativa entre os fluxos. No entanto, a inserção numa lista ordenada é uma operação cuja complexidade cresce rapidamente com o aumento do número de fluxos, pelo que a escalabilidade deste algoritmo é baixa.

Para solucionar o problema de escalabilidade do algoritmo FQ, foi desenvolvido o algoritmo SFQ [17], baseado na existência de um número de filas de espera razoavelmente elevado, sendo usada uma tabela de *hash* para seleccionar a fila em que são colocados os pacotes de um dado fluxo. As filas activas, isto é, que têm pacotes pendentes para transmissão, são servidas segundo uma disciplina *round-robin*, sendo transmitindo um pacote (ou um certo volume de dados para garantir equidade) de cada uma delas em cada passagem. Em [17] foram ainda propostos métodos eficientes para a sua implementação (com partilha de *buffers*, por exemplo). A limitação principal deste algoritmo é que quando o número de fluxos activos se aproxima do número de filas de espera ($N_{fluxos\,activos} \geq \frac{1}{10}$ ou $\frac{1}{5} N_{filas\,de\,espera}$) começa a não ser desprezável a probabilidade de colisão dos valores de *hash*, o que prejudica a equidade do algoritmo[8]. Isto pode evitar-se através de um dimensionamento correcto. Além disso, este algoritmo é eficiente e escalável em termos de

[8]Fluxos em que haja colisão do valor de *hash* são tratados como se fossem um só, partilhando o débito disponível.

processamento.

Embora não sejam utilizáveis na diferenciação de serviços, estes algoritmos, principalmente o SFQ, dada a sua eficiência, podem ser usados para garantir uma distribuição equitativa de recursos entre fluxos pertencentes a classes com tratamento tipo melhor esforço.

3.8 RED, RIO, WRED e GRED

A família de algoritmos RED (*random early detection*) foi desenvolvida tendo em vista a redução do tamanho médio das filas de espera nos nós de comutação, evitando a ocorrência de congestionamento nos nós, com a implícita perda dos pacotes que no instante de chegada ao nó encontram a fila cheia. Esta eliminação no fim da fila, além de inerentemente injusta, pode provocar um efeito designado sincronização global, que consiste na actuação simultânea dos mecanismos de controlo de congestionamento dos protocolos de transporte dos vários fluxos, e que cria uma situação em que o tráfego oscila entre um débito excessivo originando perdas e um débito inferior àquele que a rede suportaria. O algoritmo RED simultaneamente impede o efeito de sincronização global, evita o congestionamento e faz uma distribuição justa das perdas, penalizando os fluxos com maior ocupação da fila de espera.

O funcionamento do RED tira partido do mecanismo de controlo de congestionamento do protocolo de transporte que leva a uma redução do débito em caso de perdas. O seu funcionamento é o seguinte: é calculado o comprimento médio da fila de espera; sempre que este exceda um dado nível, $\min_{th}$, é dada uma certa probabilidade de perda aos pacotes que chegam ao nó. Esta probabilidade vai aumentado linearmente até atingir um valor $P_{\max} < 1$ quando a fila atinge o comprimento médio $\max_{th}$, sendo eliminados todos os pacotes sempre que o comprimento médio da fila exceda esse valor. Porque a probabilidade de perda é constante, são eliminados mais pacotes dos fluxos mais agressivos na transmissão, pelo que a distribuição de perdas é justa. O facto de se calcular um valor médio do comprimento da fila[9] em vez do seu valor instantâneo permite absorver pequenas rajadas transitórias. Eventuais indícios de ocorrência de congestionamento são detectados atempadamente quando a fila começa a crescer, e controlados antes que esse congestionamento se torne problemático.

O tratamento equitativo de fluxos é contraditório com a diferenciação de serviços. Assim, num ambiente de serviços diferenciados não pode ser usado o algoritmo RED na sua forma original. Uma variante deste algoritmo designada RIO (*RED with In/Out bit*) permite diferenciar a probablidade de perda entre pacotes conformes (*in profile*) e não conformes (*out of profile*), existindo dois mecanismos RED com diferentes parâmetros para cada um destes grupos de pacotes. Através de uma boa configuração destes parâmetros o RIO permite que comecem a ser eliminados pacotes não conformes assim que existam indícios de congestionamento ligeiro, eliminando pacotes conformes apenas em caso de congestionamento grave. Uma outra variante do RED, designada WRED (*weighted* RED), usa uma estratégia semelhante ao RIO mas em lugar do bit *In/Out* usa o campo

[9] Este valor médio é calculado submetendo o valor instantâneo a uma função passa-baixo, e.g. EWMA. Dependendo da implementação, pode ser ou não possível parametrizar esta função.

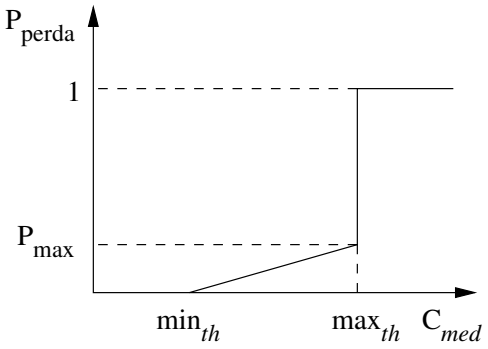

Figura 3.2: Algoritmo RED

de precedência do cabeçalho IP para diferenciar probabilidades de perda. Em [18] é apresentada uma terceira variante do RED, designada GRED (*generalized* RED), baseada na existência de n conjuntos de parâmetros RED, cada uma com uma fila virtual[10] associada. O conjunto de parâmetros para cada pacote é seleccionado com base num índice, `tc_index`, dependente do resultado de uma classificação do pacote, que pode basear-se nos mais diversos parâmetros. Assim, tanto o RIO como o WRED podem considerar-se casos particulares do GRED. Esta última variante é particularmente útil na implementação do grupo de PHB *Assured Forwarding*, apresentado na secção 2.6.2.

[10]Fisicamente existe apenas uma fila para evitar o reordenamento de pacotes.

Capítulo 4

Ambiente de desenvolvimento e teste

O ambiente de denvolvimento e teste utilizado beseia-se em computadores pessoais (PC) com o sistema operativo Linux, alguns dos quais configurados como *routers* e outros como equipamento terminal. Todos os PC utilizados dispõem de placas de rede ATM, o que permite uma grande flexibilidade na configuração, uma vez que os troços em teste são circuitos virtuais CBR com débito configurável.

4.1 Controlo de tráfego em Linux

O núcleo (*kernel*) do sistema operativo Linux inclui um conjunto bastante rico de funções de controlo de tráfego. Aliado ao facto de essas funções estarem organizadas segundo uma arquitectura extremamente flexível, isto faz do Linux a plataforma de eleição para efectuar trabalho experimental na área da qualidade de serviço.

A arquitectura de controlo de tráfego no Linux baseia-se em quatro conceitos-chave:

- Disciplinas de serviço
- Classes
- Filtros
- Policiamento

A cada interface de rede está associada uma disciplina de serviço. As disciplinas de serviço mais complexas contêm classes, podendo recorrer a filtros para atribuir cada pacote a uma das classes. Embora a existência de classes e a sua semântica estejam intimamente ligadas à disciplina de serviço, é possível combinar de forma arbitrária o uso de filtros com as disciplinas de serviço e classes, desde que estas existam.

Uma particularidade que torna o controlo de tráfego em Linux extremamente flexível é que, em princípio, as classes não são responsáveis por manter filas de pacotes: elas usam uma outra disciplina de serviço (disciplina de serviço interior) para o fazer. Esta recursividade entre disciplinas de serviço

e classes permite criar serviços com semântica elaborada a partir de componentes relativamente simples.

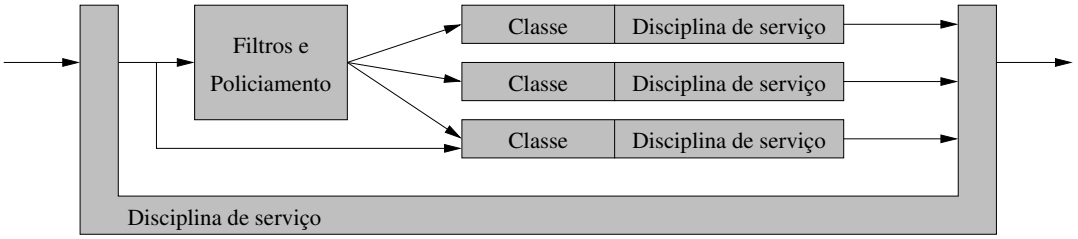

Figura 4.1: Arquitectura de controlo de tráfego em Linux

O policiamento pode ser usado na colocação de pacotes em filas para efectuar determinadas acções sobre fluxos que excedam um dado débito. É de salientar que as acções de policiamento possíveis não se limitam à eliminação de pacotes, podendo ser, por exemplo, a atribuição de pacotes a classes com qualidade de serviço inferior.

4.1.1 Disciplinas de serviço e classes

Tanto as disciplinas de serviço como as classes são identificadas, do ponto de vista do utilizador, por inteiros de 32 *bits* estruturados em duas partes de 16 *bits* cada, segundo a forma superior:inferior (*major*:*minor*), em que a parte superior é sempre diferente de zero. As instâncias de disciplinas de serviço distinguem-se das classes pela parte inferior do seu identificador, que no caso das disciplinas de serviço é zero, e é diferente de zero no caso das classes. A parte superior do identificador de uma classe deve ser igual à da disciplina de serviço que a contém. Internamente, o núcleo utiliza um outro identificador para as classes, o ID interno (*internal* ID). Este identificador é do tipo `unsigned long`[1], sempre diferente de zero, e único dentro de uma disciplina de serviço. É, no entanto, possivel que múltiplos identificadores de classe sejam mapeados no mesmo ID interno. Quando isto acontece, o identificador de classe fornece informação adicional relativamente à classificação.

Quando um pacote é passado a uma disciplina de serviço é, normalmente, invocada uma função de classificação. Esta vai devolver uma estrutura que inclui o identificador de classe (e, possivelmente, o ID interno) e também, eventualmente, decisões relativas ao policiamento.

Para tráfego gerado localmente, existe um campo no cabeçalho interno de cada pacote[2] que pode ser usado como um atalho na classificação: o `skb->priority`. Se este contiver um identificador de classe, o pacote é imediatamente atribuído a essa classe, estando isento de posterior classificação. O valor deste campo é retirado de um campo da *socket* pela qual o pacote foi enviado, o `sk->priority`, que pode ser especificado através da opção `SO_PRIORITY`. É de salientar que o campo `skb->priority` pode conter outros tipos de prioridade, como o extraído do campo TOS do

[1] Inteiro não negativo de 32 *bits*.
[2] Por "cabeçalho interno" entenda-se a estrutura usada pelo núcleo para identificar e tratar os pacotes.

cabeçalho IP. No entanto, todos estes valores são sempre inferiores ao menor identificador possível para uma disciplina de serviço ou classe, que é 65536, ou 1:0 na notação superior:inferior.

Estando o pacote atribuído a uma dada classe, é invocada a função para o passar à disciplina de serviço interior, repetindo-se o processo até a disciplina de serviço encontrada não dispor de classes.

As disciplinas de serviço encontram-se organizadas numa estrutura hierárquica. Cada interface de rede possui a sua própria hierarquia, em que a disciplina principal é identificada como `root`, e as restantes pela classe em que estão instaladas (`parent`). Existe ainda uma outra hierarquia, `ingress`, que permite controlar os pacotes à entrada na interface. Esta hierarquia não dispõe de toda a funcionalidade da hierarquia normal de saída, e neste momento apenas é possível o uso da disciplina de serviço especial `ingress`. Juntamente com filtros e, eventualmente, estimadores, pode usar-se para policiar fluxos de tráfego que chegam por essa interface. Isto permite oferecer uma funcionalidade semelhante ao *Commited Access Rate* presente em algum equipamento da Cisco Systems.

4.1.2 Filtros

Os filtros são usados pelas disciplinas de serviço para atribuir cada pacote a uma das suas classes. Este procedimento é efectuado no momento em que o pacote é passado à disciplina de serviço.

Os filtros são organizados em listas, existindo uma lista por disciplina de serviço ou classe, dependendo da disciplina de serviço. Dentro destas listas, os filtros distinguem-se entre si através de dois parâmetros: o protocolo a que se aplicam e a prioridade do filtro. Por este motivo, filtros na mesma lista e para o mesmo protocolo devem obrigatoriamente ter prioridade diferente. Na classificação de um pacote os filtros na lista são percorridos por ordem de prioridade.

Internamente os filtros podem ser estruturados em elementos. Cada elemento de um filtro é identificado por um *handle* de 32 *bits*. O *handle* 0 refere-se sempre ao próprio filtro.

Os filtros podem ser de dois tipos: genéricos ou específicos. Os filtros genéricos vão buscar o identificador de classe ao descritor do pacote, onde foi colocado por qualquer outra entidade da pilha protocolar, e.g. a camada de *firewall*. Uma única instância de um filtro genérico pode, assim, seleccionar pacotes para todas as classes. Ao contrário, os filtros específicos apenas podem seleccionar uma classe, pelo que é necessário uma instância do filtro ou um elemento do filtro por cada classe. Uma vez que existe pelo menos uma instância de um filtro específico ou de um seu elemento por classe, é possível a este tipo de filtro armazenar o ID interno da classe, poupando à disciplina de serviço o trabalho de fazer a conversão do identificador de classe para o ID interno. Nos filtros genéricos, no entanto, isto não é possível.

4.1.3 Estimadores

Os estimadores são blocos que podem ser utilizados quer pelas disciplinas de serviço quer pelos filtros para avaliar o débito médio de um dado fluxo de tráfego, D_{med}, cabendo à disciplina de serviço ou ao filtro actuar em conformidade com o valor obtido. Essa avaliação é feita com base

numa média móvel de tipo exponencial (EWMA). Para configurar um estimador são fornecidos dois parâmetros: o intervalo de medida e a constante de tempo. O volume de tráfego do fluxo, D_{inst}, é avaliado periodicamente, segundo o intervalo de medida especificado. Este valor é usado para calcular a média segundo a fórmula

$$D_{med.n} = (1 - W) \times D_{med.n-1} + W \times D_{inst}$$

em que o parâmetro de peso, W, é obtido a partir da relação entre a constante de tempo e o intervalo de medida especificados.

4.1.4 Policiamento

Assumindo, por uma questão de simplicidade, uma definição relativamente lata de policiamento como qualquer acção dependente de alguma forma do volume de tráfego, podemos considerar que existem quatro tipos distintos de mecanismos de policiamento no núcleo do sistema operativo Linux: decisões de policiamento efectuadas pelos filtros; recusa da disciplina de serviço de aceitar um pacote; eliminação de um pacote de uma disciplina de serviço interior; e eliminação de um pacote na aceitação de um outro.

A função de classificação, invocada pela disciplina de serviço, pode retornar um valor que indique uma decisão de policiamento. Assim, um valor de retorno `TC_POLICE_OK` indica que o pacote foi seleccionado pelo filtro e não é necessária nenhuma acção de policiamento; um valor de `TC_POLICE_RECLASSIFY` indica que o pacote foi seleccionado pelo filtro mas, por exceder certos limites, deve ser reclassificado, em princípio para uma classe de serviço inferior; o valor `TC_POLICY_SHOT` indica que o pacote foi seleccionado mas, por violar certos limites, deve ser eliminado; por último, o valor `TC_POLICY_UNSPEC` indica que o pacote não foi seleccionado pelo filtro. A acção a tomar perante este resultado depende da implementação da disciplina de serviço: a disciplina PRIO, por exemplo, ignora decisões de policiamento.

Os parâmetros configuráveis para este primeiro tipo de policiamento são idênticos à disciplina de serviço TBF, que será abordada mais à frente, e que na realidade corresponde a um duplo *token-bucket*: os parâmetros `rate` e `burst` controlam o débito médio e os parâmetros `peakrate` e `mtu` controlam o débito de pico durante uma rajada do *token-bucket* principal. Além destes parâmetros é também configurável o valor retornado se o pacote não for conforme, que indica a acção a tomar por parte da disciplina de serviço.

Em alternativa a uma parametrização do tipo *token-bucket* é possível usar um estimador para avaliar o débito médio segundo uma função exponencial (EWMA). Neste caso, em vez dos parâmetros referidos, é utilizado o parâmetro `avrate`, que indica um valor de débito a partir do qual deve ser aplicada a acção de policiamento.

O segundo tipo de policiamento consiste na eliminação de um pacote por iniciativa da disciplina de serviço no momento em que este lhe é passado. Se a disciplina de serviço for interior, pode fazer uma chamada a uma função, `reshape_fail`, da disciplina de serviço que a contém, se esta

a disponibilizar. Esta pode efectuar uma reclassificação para uma classe correspondente a uma qualidade de serviço inferior. Na presente implementação, no entanto, apenas a disciplina de serviço CBQ disponibiliza esta função, e apenas as disciplinas FIFO e TBF fazem uso dela. Nos restantes casos o pacote é simplesmente eliminado.

O terceiro tipo de policiamento ocorre quando uma disciplina de serviço pretende eliminar um pacote de uma outra disciplina interior. Isto é feito através da chamada à função `drop`, que a disciplina interior deve implementar. Este mecanismo é usado, nomeadamente, pela disciplina de serviço CBQ para limitar classes que estajam a transmitir acima do seu limite.

O quarto e último mecanismo de policiamento consiste na eliminação, por parte da disciplina de serviço, de um pacote que já está na fila quando um outro de maior importância lhe é passado.

4.1.5 Pacote diffserv

O pacote *linux-diffserv* adiciona alguma funcionalidade necessária para implementar o modelo de Serviços Diferenciados recorrendo ao suporte de controlo de tráfego do núcleo do sistema operativo Linux. As alterações efectuadas por este pacote são essencialmente quatro.

A primeira alteração é a adição de um novo campo, `tc_index`, aos descritores internos de pacotes. Este campo serve para manter informação sobre a classificação do pacote ao longo do seu percurso pelas disciplinas de serviço até à sua transmissão. Sem esta informação o pacote poderia ter que ser reclassificado por diversas vezes. Além disso, o facto de a classificação poder depender de factores como o tempo, tornaria a reclassificação não apenas pesada, mas mesmo impossível.

Outra alteração é a adição de uma pseudo-disciplina de serviço, `dsmark`. Esta tem duas funções: uma é obter o DSCP (*differentiated services codepoint*) dos pacotes quando estes lhe são passados e a colocação desse resultado no referido campo `tc_index`; outra é a marcação do DSCP à saída.

A terceira alteração feita pelo pacote *linux-diffserv* é a adição de um filtro, `tcindex`, que usa o campo `tc_index`, ou parte dele, para classificar os pacotes.

Por fim, a quarta alteração introduzida por este pacote é a adição de um mecanismo que possibilite a configuração de diferentes prioridades de perda, conforme é requerido pelo PHB (*per-hop behaviour*) *Assured Forwarding*. Este mecanismo é a pseudo-disciplina de serviço GRED, cujo algoritmo foi descrito na secção 3.8. A selecção da fila virtual, à qual está associada uma curva RED, é feita com base nos *bits* menos significativos do campo `tc_index`.

As versões recentes do núcleo Linux (versões preliminares de teste do núcleo 2.4.0) já incluem toda esta funcionalidade. A estas versões o pacote *linux-diffserv* apenas adiciona alguma informação e alguns *scripts* básicos de configuração.

4.2 Configuração de QoS em Linux

A configuração dos mecanismos de controlo de tráfego no Linux é feita pelos programas a nível de utilizador usando *sockets* da família AF_NETLINK. Este procedimento é, no entanto, complexo, pelo que a configuração é, normalmente, feita através da linha de comando usando uma ferramenta,

`tc`, que interpreta um conjunto de parâmetros humanamente inteligíveis e, após eventual conversão desses parâmetros, comunica com o núcleo através das referidas *sockets*. Nesta secção é abordada a configuração dos componentes de controlo de tráfego usando esta ferramenta.

Antes ainda de uma abordagem à configuração dos mecanismos de QoS em Linux, é conveniente referir alguns factores de ordem prática. O primeiro é que neste sistema operativo não existe interrupção de fim de transmissão de pacote. Em certas circunstâncias isto pode levar a um comportamento sub-óptimo das disciplinas de serviço. Se a ligação estiver permanentemente carregada, pode assumir-se que o fim de transmissão de um pacote é indicado pela transmissão do seguinte. No entanto, se houver períodos de tempo em que não existe tráfego este pressuposto não é válido, pelo que tem que existir uma estimativa.

Outro factor que pode afectar os mecanismos de controlo de tráfego é o de normalmente as placas de rede e/ou os respectivos controladores (*drivers*) fazerem uso de *buffers* internos, por questões de eficiência. O intervalo de tempo entre a passagem do pacote ao controlador de rede e o verdadeiro início da transmissão pode afectar o desempenho real das disciplinas de serviço.

Um terceiro factor a ter em conta é que, ao contrário do que normalmente acontece no âmbito das telecomunicações e redes de dados, para a ferramenta `tc` 1 Kbit significa 1024 *bits*, e 1 Mbit 1024^2 *bits*. Pode, portanto, ser necessário efectuar algumas conversões. Além disso, os mesmos identificadores são usados na configuração de débitos, significando o volume de dados referido por segundo. É também possível configurar débitos em octetos por segundo em vez de *bits* por segundo. Neste caso usa-se bps, Kbps ou Mbps ($1\,\text{Mbit} = 128\,\text{kbps}$)[3]. As convenções aqui usadas diferem, pois, da prática comum.

Volume de dados	
b	1 octeto
kb	1024 octetos
mb	1024^2 octetos
kbit	1024 *bits*
mbit	1024^2 *bits*

Débito ou capacidade	
bps	octetos/s
kbps	1024 octetos/s
mbps	1024^2 octetos/s
kbit	1024 *bits*/s
mbit	1024^2 *bits*/s

Tabela 4.1: Convenções para a ferramenta `tc`

Por fim, a maior parte das opções passadas ao comando `tc` podem ser abreviadas até ao comprimento mínimo que as torne unicamente identificáveis. Por exemplo, a opção geral `-statistics` pode ser abreviada para `-s`, pois não existe nenhuma outra cujo início seja `-s`. Por outro lado, algumas opções podem ser invocadas usando diferentes identificadores, permitindo a cada utilizador usar o que lhe é mais familiar ou lhe parece mais descritivo.

O resto desta secção contém uma descrição detalhada da configuração dos mecanismos de QoS do Linux a partir da linha de comando, com a ferramenta `tc`. A maior parte da informação nela contida provém essencialmente da análise do código-fonte da ferramenta `tc` e da parte relativa ao controlo de tráfego do núcleo do Linux, pelo que inclui alguns parâmetros (até agora) não

[3]A interpretação destes campos não distingue maiúsculas e minúsculas, pelo que não se pode aqui usar a vulgar distinção entre b (*bits*) e B (octetos).

documentados.

4.2.1 tc

A sintaxe do comando `tc` é a seguinte:

```
tc [OPTIONS] OBJECT {COMMAND | help}
```

O campo `OBJECT` define o objecto que se pretende configurar. Este objecto pode ser:

`qdisc` para configuração de disciplinas de serviço

`class` para configuração de classes

`filter` para configuração de filtros

Se, no entanto, em lugar de um destes objectos for especificado `help`, é mostrada alguma informação de ajuda sobre a sintaxe básica do comando `tc`.

Os comandos possíveis para o campo `COMMAND` são específicos para cada tipo de objecto, pelo que serão referidos nas respectivas secções. O comando especial `help` serve para obter alguma informação específica para a configuração do tipo de objecto especificado. É de referir que a palavra `help` no fim de qualquer linha de comando `tc` permite, normalmente, obter ajuda o mais específica possível para o tipo de operação que se pretende efectuar.

Quanto ao campo `OPTIONS`, as opções possíveis são:

`-stats`, `-statistics` para obter estatísticas sobre o objecto especificado

`-details` para aumentar o nível de detalhe na listagem de objectos

`-raw` alguns parâmetros fornecidos numa forma humanamente legível são convertidos pelo `tc` e armazenados internamente num outro formato por razões de eficiência; esta opção serve para mostrar esses parâmetros na forma armazernada internamente, adicionalmente ào valor humanamente legível que é normalmente mostrado

`-Version` permite obter informação sobre a versão do comando `tc` em uso

`-help` mostra informação sobre a sintaxe básica do comando `tc`

4.2.2 tc qdisc

As duas formas possíveis para a sintaxe do comando `tc` para manipulação de disciplinas de serviço são as seguintes:

```
tc qdisc [COMMAND] dev STRING [handle QHANDLE]
[root|ingress|parent CLASSID] [estimator INTERVAL TIME_CONSTANT]
[[QDISC_KIND] [help|OPTIONS]]

tc [TC_OPT] qdisc [LS_COMMAND] [dev STRING] [ingress]
```

A segunda forma suporta apenas os comandos de listagem, `list`, `show` e `lst`, de significado equivalente para o `tc`. Estes comandos permitem obter uma listagem das disciplinas de serviço instaladas. As opções `TC_OPT` são as do comando `tc`, descritas na secção 4.2.1. A opção `dev`, se utilizada, limita o âmbito da listagem às disciplinas de serviço instaladas na interface especificada em `STRING`. A opção `ingress` deveria restringir a listagem à hierarquia de entrada, mas aparentemente é ignorada pelo `tc`.

A primeira forma usa-se com os restantes comandos suportados pelo `tc qdisc`:

`add`	usado para adicionar uma disciplina de serviço à hierarquia
`change`	usado para alterar parâmetros de uma disciplina de serviço instalada
`replace`	usado para substituir uma hierarquia completa, ou parte dela, pela disciplina de serviço especificada
`link`	teoricamente, teria uma função idêntica a `replace`, com a excepção de que se não existisse a disciplina de serviço a ser substituída não seria criada a nova; na prática, um problema de implementação faz com que tenha um comportamento semelhante a `change`
`delete`	usado para remover uma hierarquia de disciplinas de serviço
`help`	usado para obter ajuda sobre a utilização do comando `tc` na manipulação de disciplinas de serviço

O parâmetro `dev` serve para identificar a interface de rede sobre cuja hierarquia se pretende actuar, sendo a sua especificação obrigatória. A opção `handle` é usada para especificar o identificador a utilizar para a disciplina de serviço, na forma superior:inferior (ambos em notação hexadecimal). Como a parte inferior é sempre 0 para as disciplinas de serviço, é possível abreviar este parâmetro para superior: . Se não for passada esta opção na criação de uma disciplina de serviço, o sistema atribui-lhe automaticamente o identificador, começando em 8000: e incrementando este valor por cada operação deste tipo.

A posição da disciplina de serviço em causa é indicada usando o identificador da classe em que está (ou vai ficar) instalada, na forma `parent CLASSID`. No caso particular de disciplinas instaladas na raíz da hierarquia para a interface de rede é utilizado o identificador `root`. De forma semelhante, para uma disciplina de serviço instalada na hierarquia especial de ingresso é utilizado o identificador `ingress`.

Se a disciplina de serviço fizer uso de um estimador, este pode ser instalado usando a opção `estimator`, juntamente com os dois parâmetros necessários: o intervalo de medida e a constante de tempo da EWMA.

Por fim, são especificados o tipo de disciplina de serviço e eventuais parâmetros que necessite. Como é habitual, a opção `help` depois do tipo de disciplina de serviço permite obter ajuda sobre o modo como configurá-la, nomeadamente no que se refere a parâmetros.

Para remover uma disciplina de serviço com `tc qdisc delete` apenas é necessário especificar a interface de rede (com o parâmetro `dev`) e a classe onde está instalada (usando um dos parâmetros

`parent`, `root` ou `ingress`).

4.2.3 `tc class`

Tal como acontece nas disciplinas de serviço, existem duas formas distintas da sintaxe do comando `tc` para a manipulação de classes, que são as seguintes:

```
tc class [COMMAND] dev STRING [classid CLASSID] [parent CLASSID]
[[QDISC_KIND] [help|OPTIONS]]

tc [TC_OPT] class [LS_COMMAND] dev STRING
```

Tal como nas disciplinas de serviço, a segunda forma suporta apenas os comandos de listagem, `list`, `show` e `lst`, de significado equivalente. As opções `TC_OPT` são as descritas na secção 4.2.1. O parâmetro `dev` é obrigatório neste caso[4], indicando a interface de cuja hierarquia se pretende obter informação relativa a classes.

A primeira forma usa-se para os restantes comandos, que são os seguintes:

- `add` — usado para adicionar uma classe à hierarquia; nem sempre isto é necessário, uma vez que certas disciplinas de serviço com classes, e.g. `prio`, instalam automaticamente todas as classes possíveis
- `change` — usado para alterar parâmetros de uma classe instalada, se aplicável
- `replace` — tal como `change` pode ser usado para alterar parâmetros de uma classe instalada, com a diferença que a classe é criada se não existir
- `delete` — usado para remover uma classe e, consequentemente, a parte da hierarquia que dela descende
- `help` — permite ober informação sobre a utilização do comando `tc` na manipulação de classes

A interface de rede sobre cuja hierarquia se pretende actuar é especificada pelo uso do parâmetro `dev`. O parâmetro `parent` identifica a disciplina de serviço à qual a classe pertence. A opção `classid` serve para atribuir um identificador à classe, se estiver a ser adicionada, ou para a identificar, se apenas se pretender alterar-lhe parâmetros. Uma vez que a parte superior do identificador é necessariamente igual à da disciplina de serviço à qual pertence, é possível omiti-la, abreviando o identificador para :inferior. Se o identicador for omitido na criação da classe, é-lhe atribuído um automaticamente pelo sistema, que se inicia em :8000 e é incrementado por cada identificador automaticamente atribuído. `QDISC_KIND` identifica o tipo de disciplina de serviço que contém a classe, e é usado para interpretação de parâmetros ou opções específicas incluídas no campo `OPTIONS`. Este parâmetro é obrigatório, excepto para a remoção. A opção `help` fornece informação sobre a manipulação de classes no tipo de disciplina de serviço especificado.

[4] Se não for especificada a listagem obtida é vazia.

Para remover uma classe apenas é necessário especificar a interface de rede (`dev`) e o identificador da classe (`classid`).

4.2.4 `tc filter`

Para a manipulação de filtros, à semelhança do que acontece nos dois casos anteriores, existem duas alternativas para a sintaxe do comando `tc`:

```
tc filter [COMMAND] dev STRING [parent CLASSID] [preference PRIO]
[protocol PROTO] [estimator INTERVAL TIME_CONSTANT ] [handle FILTERID]
[[FILTER_TYPE] [help | OPTIONS]]
```

```
tc [TC_OPT] filter [LS_COMMAND] dev STRING [parent CLASSID]
```

Tanto as opções `TC_OPT` como os comandos `LS_COMMAND` possíveis na segunda forma são em tudo idênticos aos casos anteriores. Tal como na listagem de classes, o parâmetro `dev` é obrigatório[5]. A opção `parent` serve para limitar a listagem de filtros aos instalados na disciplina de serviço identificada por `CLASSID`.

Na primeira forma os comandos possíveis são os seguintes:

`add`	usado para adicionar um novo filtro ou um elemento a um filtro existente
`change`	usado para modificar parâmetros de filtros ou elementos de filtros instalados[6]
`replace`	usado para substituir um filtro
`delete`	usado para remover um filtro ou um elemento de filtro instalados
`help`	para obter informação sobre a manipulação de filtros com o comando `tc`

A interface de rede é especificada através do argumento do parâmetro `dev`. A opção `parent` identifica a disciplina de serviço ou classe em que o filtro está instalado. A prioridade do filtro, usada quer na sua identificação quer na determinação da ordem de aplicação dos diversos filtros (filtros com valor de prioridade mais baixo são avaliados primeiro) é especificada pela opção `preference` (que também pode ser designada `priority`). A opção `protocol` serve para especificar o protocolo ao qual o filtro se aplica. A opção `estimator` é usada para instalar um estimador, de modo a permitir ao filtro desempenhar funções de policiamento. A opção `handle` permite identificar o elemento do filtro a manipular, sendo o seu formato dependente do tipo de filtro, especificado em `FILTER_TYPE`. Por fim podem passar-se opções específicas do filtro, se necessário.

Na instalação de um filtro o parâmetro `protocol` é obrigatório; se `preference` for omitido o `tc` assume o valor 0xC000 (49152 decimal); se `parent` for omitido, o sistema assume que é a disciplina de serviço instalada na raiz da hierarquia. Na remoção de filtros o parâmetro `preference` é obrigatório; se se omitir `parent`, é usada a disciplina de serviço instalada na raiz.

[5] Se não for especificado, a listagem obtida é vazia.

[6] Com as versões do núcleo Linux e da ferramenta `tc` utilizados, os comandos `change` e `replace` para filtros não funcionam da forma esperada. No entanto, qualquer configuração pode ser obtida apenas com os comandos `add` e `delete`.

4.2.5 `estimator`

Para instalar um estimador nos filtros ou disciplinas de serviço que o suportem é usada a seguinte sintaxe:

```
estimator INTERVAL TIME_CONSTANT
```

Ambos os parâmtros podem ser especificados em segundos (`secs`), milissegundos (`msecs`) ou microssegundos (`usecs`). Internamente, o intervalo de medida é arredondado para cima e armazenado como potência de 2, com valores válidos entre -2 e 3, pelo que os valores que assume podem variar entre 250 milissegundos e 8 segundos. O parâmetro W (ver secção 4.1.3) é calculado por forma a que a constante de tempo efectiva, T, seja a seguinte:

$$T = \frac{A}{-\ln(1-W)}$$

em que A é o valor efectivo do intervalo de medida. T é o valor arredondado para baixo da constante de tempo especificada, tal que W seja uma potência inteira de 2, sendo armazenado internamente apenas o expoente, `ewma_log`. Uma vez que os valores válidos de `ewma_log` variam entre 1 e 30, os valores possíveis para a constante de tempo variam entre 1.44 e $1.07e9$ vezes o intervalo de medida efectivo.

4.2.6 `police`

Para instalar um bloco de policiamento num filtro usa-se a seguinte sintaxe para o parâmetro `police`:

```
police rate BPS burst BYTES[/BYTES] [mtu BYTES[/BYTES]]
[peakrate BPS] [mpu BYTES] [avrate BPS] [ACTION[/ACTION]]
```

O parâmetro `rate` define o débito médio do *token-bucket* principal, e o parâmetro `burst` (que também pode ser designado `buffer` ou `maxburst`) a sua profundidade. O segundo valor, opcional, do parâmetro `burst` determina o tamanho das células na tabela de busca interna[7], devendo ser uma potência inteira de 2. Se não for especificado, é calculado o valor óptimo, que é o máximo para o qual $\frac{\text{mtu}}{\text{célula}} < 256$. Se o parâmetro `mtu` não for especificado, assume-se 2047 para o cálculo da célula. Em interfaces cuja MTU seja superior a 2040[8] é obrigatória a especificação do parâmetro `mtu`, sob pena de serem eliminados todos os pacotes de tamanho superior a este. Mesmo em interfaces em que isto não aconteça é conveniente especificá-lo para garantir a granularidade mais fina possível no cálculo das tabelas.

[7] Por razões de eficiência de implementação, são usadas tabelas pré-calculadas de tempos de transferência de pacotes. Estas tabelas têm no máximo 256 entradas, designadas células. Assim, diversos tamanhos de pacote próximos (em número igual ao tamanho da célula) são considerados como tendo idênticos tempos de transferência.

[8] Para o MTU padrão de 2047, a célula é 8. Para efeitos de policiamento, se não for especificado o MTU é usado o valor 255 × célula, conduzindo a um valor de 2040 octetos neste caso.

O parâmetro `peakrate` define o débito máximo do *token-bucket* de controlo de débito em rajada e activa-o. Se `peakrate` for especificado, o parâmetro `mtu` (que também pode ser designado `minburst`), além da função referida, controla a profundidade deste *token-bucket*. Neste caso, o parâmetro `mtu` deve ser inferior ao parâmetro `burst`.

O parâmetro `mpu` permite configurar a unidade mínima de policiamento, isto é, qualquer pacote de tamanho inferior é considerado, para efeitos de policiamento, como tendo esse tamanho.

Em alternativa ao (duplo) *token-bucket* é possível usar o débito médio, obtido através de uma EWMA, para efeitos de policiamento. Para tal, é necessário instalar um estimador no filtro. O bloco de policiamento é então configurado com o parâmetro `avrate`, que indica o limite a partir do qual o tráfego se considera excedente.

O último parâmetro permite especificar a acção a tomar para tráfego em excesso. As acções possíveis são:

`continue` para passar o controlo ao filtro seguinte, eventualmente fazendo novo policiamento; isto permite passar o tráfego excedente de classes de elevada prioridade para classes de prioridade inferior

`drop, shot` para eliminar o pacote

`pass, ok` para aceitar o pacote

`reclassify` para informar a disciplina de serviço que o pacote é excedente, cabendo a esta reclassificá-lo

É também possível configurar uma segunda acção, que será aplicada aos pacotes não excedentes. Esta facilidade, no entanto, não é normalmente utilizada por ser de pouco interesse, uma vez que a acção padrão para estes pacotes é a aceitação.

Existe ainda um parâmetro escondido, `index`, que permite passar um índice entre 0 e 15 para a tabela de *hash* usada internamente no armazenamento e busca de blocos de policiamento. Normalmente esta atribuição é automática, pelo que o utilizador não precisa de se preocupar com ela.

4.2.7 Disciplinas de serviço e classes

Genérica

Sempre que não está explicitamente configurada uma disciplina de serviço na raiz de uma hierarquia (`root`), os pacotes são servidos pela disciplina de serviço genérica, também designada `pfifo_fast`[9].Esta consiste num escalonador de prioridade estrita com três bandas (filas). Dentro de cada banda, os pacotes são escalonados segundo o princípio FIFO.

A disciplina de serviço genérica é completamente transparente para o utilizador, o que pode comprovar-se executando `tc qdisc ls` numa interface em que não tenha sido instalada nenhuma

[9]Em algumas interfaces virtuais como a de *loopback* (`lo`) ou a `dummy` (que elimina os pacotes por ela enviados) a disciplina de serviço pode ser `noop` ou `noqueue`, mas em interfaces reais é sempre a genérica, `pfifo_fast`.

disciplina de serviço[10]. Por este motivo, a disciplina de serviço genérica não possui classes (as filas são internas à própria disciplina de serviço), nem pode fazer uso de filtros.

A selecção de banda para cada pacote é feita com base nos 4 *bits* menos significativos do campo `skb->priority` do cabeçalho interno do pacote. Como foi referido na secção 4.1.1, o valor deste campo pode ser explicitamente indicado para tráfego gerado localmente. Quando isto não acontece (se não for usada a opção da *socket* ou para tráfego gerado externamente) é colocado neste campo um valor entre 0 e `TC_PRIO_MAX` (15) obtido a partir do campo TOS extraído do octeto TOS do cabeçalho IP[11]. Os valores definidos[12] são:

```
TC_PRIO_BESTEFFORT        0
TC_PRIO_FILLER            1
TC_PRIO_BULK              2
TC_PRIO_INTERACTIVE_BULK  4
TC_PRIO_INTERACTIVE       6
TC_PRIO_CONTROL           7

TC_PRIO_MAX               15
```

O mapeamento entre os valores possíveis e a banda é fixo: {1, 2, 2, 2, 1, 2, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1}[13]. Por exemplo, é por este motivo que mesmo sem configurar qualquer mecanismo de qualidade de serviço os pacotes contendo o valor `LOW_DELAY` no campo TOS do octeto TOS do cabeçalho IP, mapeados para o valor `TC_PRIO_INTERACTIVE` e, portanto, para a banda 0, são tratados prioritariamente.

Dadas as limitações inerentes, a disciplina de serviço genérica é computacionalmente mais eficiente que a combinação `prio` + `pfifo`, que na sua configuração-padrão produz resultados semelhantes.

fifo

Na realidade existem não uma mas duas disciplinas de serviço FIFO, que se distinguem entre si pela forma como é feita a limitação de pacotes na fila. Assim, enquanto a disciplina `pfifo` limita o número de pacotes que a fila pode conter, a disciplina `bfifo` limita o número total de octetos de todos os pacotes na fila.

A sintaxe para a instalação das disciplinas `pfifo` ou `bfifo` com a ferramenta `tc` é a seguinte:

```
(p|b)fifo [limit NUMBER]
```

[10] No entanto pode verificar-se a sua presença usando o comando `ip addr show ITF`, em que ITF é a interface de rede em causa.

[11] O mapeamento entre o campo TOS e o valor de `skb->priority` está contido no vector `ip_tos2prio[]`, definido em `linux/net/ipv4/route.c`.

[12] `linux/include/linux/pkt_sched.h`

[13] `linux/net/sched/sch_generic.c`

O parâmetro `limit` é usado para indicar o limite em octetos (`bfifo`) ou em número de pacotes (`pfifo`) da fila. Se este parâmetro for omitido, assume-se igual ao limite padrão do número de pacotes para a interface (`pfifo`) ou este número multiplicado pelo MTU da interface (`bfifo`).

tbf

A disciplina de serviço `tbf` implementa um algoritmo de duplo *token-bucket*, para controlo do débito médio e, simultaneamente, do débito de pico. Este algoritmo é semelhante ao usado em `police`, com a diferença que o objectivo aqui é atrasar os pacotes que excedam a especificação, e não eliminá-los. A sintaxe para a instalação da disciplina de serviço `tbf` é a seguinte:

```
tbf (limit BYTES | latency TIME) rate BPS burst BYTES[/BYTES]
[mtu BYTES[/BYTES] [peakrate BPS]] [mpu BYTES]
```

Tal como em `police` (ver secção 4.2.6), o parâmetro `mtu` é usado para calcular as células quer para a tabela de débito de pico quer para a tabela de débito médio, sendo recomendável (e mesmo obrigatório, se a MTU da interface for superior a 2040) especificá-lo. O valor indicado de `mtu` deve incluir o cabeçalho da camada de ligação lógica. O parâmetro `burst`, que também pode designar-se `buffer` ou `maxburst`, especifica a profundidade do *token-bucket* principal, ou seja, o número máximo de octetos que é possível transmitir em rajada após um período de inactividade. Deve ser sempre maior ou igual ao parâmetro `mtu`. O parâmetro `rate` especifica o débito médio obtido em regime permanente.

O parâmetro `limit` serve para especificar o tamanho da fila em octetos. Deve ser maior ou igual à MTU da interface (neste caso excluindo o cabeçalho de nível 2) sob pena de pacotes de tamanho superior ao aqui especificado serem eliminados. Em alternativa, pode ser usado o parâmetro `latency`, indicando o atraso máximo sofrido por um pacote na fila. Neste caso o tamanho da fila é calculado internamente segundo a fórmula $\texttt{limit} = \texttt{rate} \times \texttt{latency} + \texttt{burst}$.

O parâmetro opcional `peakrate` activa o *token-bucket* de controlo de débito de pico, e permite especificar o seu valor médio. Neste caso, o parâmetro `mtu` (ou `minburst`) indica também a profundidade deste *token-bucket* secundário.

O parâmetro `mpu` tem o mesmo significado que em `police`.

Sobre a disciplina `tbf` há a salientar uma limitação: quando um pacote é atrasado por exceder a especificação é activado um temporizador para despoletar o envio desse pacote; no entanto, a resolução do temporizador é bastante grosseira, pois é igual a 1/HZ segundos, em que HZ é uma variável interna do núcleo Linux, correspondente à frequência de comutação de processos; isto implica que se não ocorrer nenhuma interrupção (por fim de transmissão ou chegada de um pacote) a disciplina de serviço fica esse intervalo de tempo sem transmitir. Isto limita o débito de um *token-bucket* de profundidade B a $D_{crit.} = B * \mathrm{HZ}$.

O valor padrão para HZ é de 100 em plataformas x86, embora possa ser alterado para 1024 em processadores recentes desta família (ver apêndice A)[14]. Resolvendo a expressão em ordem a B

[14] Esta alteração foi efectuada na plataforma de teste.

pode verificar-se que para controlar um débito de 10 Mbit/s a profundidade do *token-bucket* deve ser no mínimo de cerca de 13 Koctetos (1.3 Koctetos se HZ=1024). No entanto, o controlo do débito de pico é muito mais difícil: uma vez que a profundidade do *token-bucket* é igual à MTU, se esta for de 1500 octetos, o limite superior para o débito de pico que é possível controlar é de apenas 1.2 Mbit/s (12 Mbit/s se HZ=1024). Uma vez que só faz sentido controlar o débito de pico se este for superior ao débito médio, pode ser impossível controlá-lo, sendo preferível não especificar `peakrate` nestas condições.

prio

A disciplina de serviço `prio` é a mais simples que contém classes. Esta disciplina de serviço implementa um escalonador de prioridade estrita, com um número de bandas configurável, sendo a prioridade tanto maior quanto menor for o número da banda. A criação de classes é feita automaticamente durante a instalação desta disciplina de serviço, pelo que não é necessário (nem sequer possível) usar `tc class` para o fazer. Os identificadores das classes assim criadas têm as partes inferiores numeradas de :1 a :n, em que n é o número de classes. Naturalmente, em cada uma destas classes é automaticamente instalada uma disciplina de serviço genérica (`pfifo_fast`), que pode, no entanto, ser substituída por qualquer outra usando `tc qdisc replace` (ou mesmo com `tc qdisc add`, uma vez que a disciplina de serviço genérica não é vista, do ponto de vista da configuração, como uma disciplina de serviço real).

A classificação dos pacotes é feita segundo três critérios:

- Se `skb->priority` contiver o identificador de uma das classes, o pacote é atribuído à banda correspondente[15,16].
- Se não contiver, é feita a classificação usando filtros, sendo estes a atribuir o pacote a uma das bandas (classes).
- Se não estiverem instalados filtros, ou a função de classificação retornar um valor diferente de `TC_POLICE_OK`, então são usados os 4 *bits* inferiores de `skb->priority` para, através do mapeamento definido, efectuar a selecção de banda.

A sintaxe para a instalação da disciplina de serviço `prio` é a seguinte:

```
prio [bands NUMBER] [priomap P1 P2...]
```

O parâmetro `bands` serve para especificar o número de classes[17] a instalar, e, logicamente, tem que ser maior ou igual a 2. Se omitido, são criadas 3 classes, cuja parte inferior do identificador

[15] A parte inferior do identificador de classe é atribuída a partir de :1, e as bandas são numeradas a partir de 0. Assim, a correspondência entre classe e banda é .

[16] Se `skb->priority` contiver a parte superior igual à da disciplina de serviço mas a parte inferior não corresponder a nenhuma das suas classes, o pacote é atribuído à mesma banda do tráfego tipo melhor esforço, isto é, à banda correspondente à primeira entrada do mapa de prioridades.

[17] Teoricamente o número de bandas deveria significar o número de filas, cada uma correspondendo a uma classe e com uma dada prioridade. No entanto, a disciplina de serviço instalada em cada uma das classes é a `pfifo_fast`, que contém 3 filas.

é igual ao número da banda mais 1. O parâmetro `priomap`, seguido de um conjunto de até 16 números, usa-se para definir o mapeamento entre o inteiro (entre 0 e 15) contido nos 4 *bits* inferiores de `skb->priority` e a banda a que o pacote é atribuído segundo o terceiro critério. Se não forem especificados todos os 16 valores do mapeamento, os restantes assumem o seu valor-padrão: 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1. Note-se que para instalar uma disciplina `prio` com apenas duas bandas é obrigatório definir o mapa de prioridades. Se o número de bandas a instalar for superior a três é também aconselhável fazê-lo, uma vez que se isto não acontecer, a menos que a classificação seja feita com base nos dois primeiros critérios as restantes permanecem inutilizadas. É de salientar que nas bandas não utilizadas em `priomap` não é instalada uma disciplina de serviço genérica, mas sim uma pseudo-disciplina `noop`, que se limita a eliminar silenciosamente todos os pacotes. Assim, sempre que se optar por usar um `priomap` que atribua todos os pacotes à mesma banda efectuando a diferenciação com base em filtros é mandatório instalar uma disciplina de serviço como descentente de cada uma das classes, excepto da correspondente à referida banda.

Por fim, convém referir que devido a um erro de implementação (*bug*) não é possível listar as classes de uma disciplina `prio` com `tc class list`, que termina com um *core dump*.

cbq

A disciplina de serviço `cbq` é a mais flexível das presentemente implementadas no núcelo do sistema operativo Linux. Uma vez que já foi apresentada com algum detalhe na secção 3.6, serão apenas abordados a sua configuração e detalhes de implementação.

Antes de mais, é de referir que nesta implementação o nível das classes é contado a partir de 0, e não de 1 como em [15].

Para a instalação de uma instância da disciplina de serviço `cbq` usa-se a seguinte sintaxe:

```
cbq bandwidth BPS avpkt BYTES [mpu BYTES] [cell BYTES] [ewma LOG]
```

O parâmetro `bandwidth`[18] permite especificar o débito da interface, e é usado para pré-calcular uma tabela de tempos de transferência para os vários tamanhos de pacotes e o parâmetro interno do CBQ *maxidle*. O parâmetro `avpkt` exprime o tamanho médio dos pacotes a enviar, e é também usado no pré-cálculo de *maxidle*. O `mpu` é o tamanho mínimo para um pacote (e.g., 64 octetos em *ethernet*). O parâmetro opcional `cell` permite especificar a granularidade da tabela de tempos de transferência, e deve ser uma potência inteira de 2. Este parâmetro está sujeito às mesmas condições que a célula no policiamento. Se omitido, a granularidade é calculada com base num tamanho máximo de pacote igual a $\frac{3}{2}avpkt$. O parâmetro opcional `ewma` permite especificar o valor logarítmico (base 2) da constante W[19] usada no cálculo de *avgidle* (ver secção 3.6) segundo a fórmula $avgidle_{n+1} = (1 - W) \cdot avgidle_n + W \cdot idle_n$. Se omitido, é usado o valor padrão 5.

[18] Também pode designar-se `rate`. No entanto é preferível não o fazer para evitar confusão com o parâmetro `rate` das classes `cbq`.

[19] Na verdade, o seu simétrico.

Após instalada a disciplina de serviço `cbq`, é necessário criar as classes. Para o efeito, usa-se `tc class` com a seguinte sintaxe:

```
cbq bandwidth BPS rate BPS [avpkt BYTES] [maxburst PKTS] [minburst PKTS]
    [bounded] [isolated] [allot BYTES] [weight RATE] [mpu BYTES]
    [prio NUMBER] [cell BYTES] [ewma LOG] [estimator INTERVAL TIME_CONSTANT]
    [split CLASSID] [defmap MASK[/CHANGE]]
```

O parâmetro `bandwidth` é usado para especificar o débito total da interface. O parâmetro `rate` especifica o débito atribuído à classe. O parâmetro `avpkt` indica o tamanho médio dos pacotes na interface, e é usado no pré-cálculo de variáveis internas como *maxidle* e *offtime*. O parâmetro `maxburst` permite indicar o número de pacotes de tamanho médio (`avpkt`) da classe que é possível enviar após um período de inactividade, sendo usado para pré-calcular a variável interna *maxidle*[20]. O parâmetro `minburst`, usado para pré-calcular a variável interna *offtime*, permite configurar o tamanho da rajada em regime permanente de uma classe regulada pelo escalonador de partilha de ligação, significando o número de pacotes de tamanho médio que podem ser enviados antes de ser forçado o tempo de espera (*offtime*).

A opção `bounded` (por oposição a `borrow`, usada por omissão) indica que a classe não deve poder exceder o débito que lhe foi atribuído, mesmo em condições em que pelas regras de partilha da ligação lho permitissem. A opção `isolated` (por oposição a `sharing`, usada por omissão), indica que a classe deve deixar vazio o campo que indica a classe da qual descende. Assim, o seu tráfego não é contabilizado nas estatísticas das classes suas antecessoras. Na secção 5.4 são apresentados testes que permitem examinar detalhadamente o comportamento destas duas opções.

Os parâmetros `allot` e `weight` são usados para calcular o *quantum* WRR, isto é, a quantidade de dados [21] transmitida em cada passagem do algoritmo WRR, segundo a fórmula

$$quantum = allot \times \frac{weight}{avgweight}$$

onde *avgweight* é a média dos valores de `weight` de todas as classes com a mesma prioridade. Recomenda-se o uso de um valor de `allot` igual a MTU + cabeçalho MAC, e um valor de `weight` proporcional ao débito afecto à classe[22]. Se omitidos, `allot` assume o valor $\frac{3}{2}$`avpkt`, e `weight` o valor de `rate` em octetos por segundo.

O parâmetro `mpu` indica o tamanho mínimo de pacote. Pacotes menores consideram-se como tendo este tamanho. A prioridade da classe, entre 1 e 8, pode ser configurada através do parâmetro `prio`. Os parâmetros `cell` e `ewma` têm o mesmo significado que em `qdisc cbq`, com a diferença que se `cell` for omitido o seu valor é calculado com base no parâmetro `allot`. A opção `estimator` permite instalar um estimador na classe para obtenção de estatísticas.

[20] O tamanho máximo, em octetos, da rajada que é possível enviar após um período de inactividade é praticamente independente do tamanho dos pacotes, e aproximadamente igual ao produto de `maxburst` por `avpkt`.

[21] Em média, pois esta implementação usa *Deficit* WRR.

[22] Para obter o comportamento padrão do CBQ.

Os parâmetros `split` e `defmap` permitem configurar um método de classificação eficiente. O parâmetro `split` indica o nó da hierarquia onde ocorre a classificação, no qual deve estar instalado um filtro genérico. Se o resultado da classificação não for um identificador de classe, os 4 *bits* menos significativos desse resultado são usados para indexar uma tabela onde podem existir identificadores de classes que não são suas descendentes directas, sendo o pacote atribuído à classe correspondente. Evita-se assim o trabalho de classificar o pacote em todos os nós seus antecessores até chegar a uma folha. O valor `MASK` do parâmetro `defmap` permite indicar quais das 16 entradas na tabela da classe indicada em `split` devem apontar para a classe a instalar: `MASK` é uma máscara em que cada *bit* 1 indica que a respectiva entrada deve apontar para a classe que se está a instalar. O valor `CHANGE` desse parâmetro, que é opcional, é essencialmente usado na reconfiguração da classe: é também uma máscara, mas indica quais as entradas na tabela que devem ser alteradas. Se omitido, o sistema assume todos os *bits* activos, indicando que todas as entradas na tabela cujo *bit* correspondente em `MASK` esteja activo e apenas essas devem apontar para esta classe. A utilização mais simples destes parâmetros é na definição da classe para onde deve ser encaminhado o tráfego tipo melhor esforço. Isto faz-se usando os parâmetros `split M:0 defmap 1`, em que `M:0` é o identificador da instância da disciplina CBQ, no comando de instalação da classe destinada a transportar este tráfego.

Note-se que toda uma hierarquia CBQ é instalada sobre uma única instância de `qdisc cbq`: o antecessor (`parent`) de uma classe CBQ pode ser outra classe CBQ em vez da própria disciplina de serviço. No entanto, segundo as regras de gestão de classes do Linux, isto implica que todas as classes, independentemente da sua posição na hierarquia CBQ, têm a parte superior do identificador de classe igual.

Sobre esta implementação da disciplina de serviço CBQ convém ainda referir que esta suporta métodos alternativos de penalização de classes que excedam a sua quota de débito. A estratégia-padrão (`CLASSIC`) é a utilização do *offtime* nas classes-folha (ver secção 3.6). Outra alternativa (`RCLASSIC`) é aplicar o *offtime* mesmo em classes que não sejam folhas. Uma terceira alternativa (`DELAY`) é introduzir um atraso suficiente para o débito médio da classe descer abaixo do seu limite[23]. A quarta alternativa (`LOWPRIO`) consiste na alteração do valor de prioridade efectiva da classe em relação ao seu valor nominal. A quinta e última opção (`DROP`) consiste na penalização por eliminação de pacotes. Apesar de estarem implementadas, estas alternativas não são configuráveis através da ferramenta `tc`, pelo que têm um interesse essencialmente experimental.

atm

A pseudo-disciplina de serviço `atm` permite desviar pacotes seleccionados da interface para serem transferidos através de circuitos virtuais ATM. Para a instalação desta pseudo-disciplina de serviço não é requerido qualquer parâmetro.

Para a configuração de classes com `tc class` a sintaxe é a seguinte:

```
atm (pvc ADDR | svc ADDR [sap SAP]) [qos QOS] [sndbuf BYTES]
```

[23] A principal diferença é que este intervalo de tempo não é pré-calculado.

```
[hdr HEX...] [excess (CLASSID | clp)] [clip]
```

Os parâmetros `pvc` ou `svc` identificam o circuito virtual a utilizar no transporte do tráfego dessa classe. No segundo caso, é possível especificar o protocolo de nível 2 na *blli* usando o parâmetro `sap`, assumindo-se `blli:l2=iso8802` (ISO 8802/2) se omitido. O parâmetro opcional `qos` permite especificar os parâmetros de tráfego do circuito virtual. Se omitido, o sistema assume `aal5,ubr:sdu=9180,rx:none`, ou seja, camada de adaptação AAL5 e categoria de serviço UBR com SDU máxima de 9180 octetos para transmissão, e ausência de tráfego no sentido inverso. O parâmetro opcional `sndbuf` permite especificar o tamanho, em octetos, do *buffer* da *socket*. O parâmetro `hdr`, seguido de um conjunto de dígitos hexadecimais (no máximo 64 octetos), permite especificar um cabeçalho a usar para os pacotes em vez do encapsulamento padrão (LLC/SNAP). A opção `excess` exprime a acção a tomar se o filtro indicar que o pacote deve ser reclassificado (por ser excedente): se for indicado um identificador de classe, os pacotes são atribuídos a essa classe; se for indicado `clp`, os pacotes são enviados através da própria classe, mas com o campo CLP=1 nas células constituintes do pacote, sendo esta última a acção padrão. A opção `clip` permite que tramas que cheguem pelo circuito virtual (que normalmente é só de saída) sejam passadas à pilha TCP/IP para processamento. Deve, portanto, ser usada sempre que o circuito virtual seja bidireccional.

sfq

A disciplina de serviço `sfq` permite servir em *round-robin* os fluxos que a atravessam, pretendendo por eles distribuir de forma quase equitativa[24] o débito disponível. Por fluxo entende-se neste âmbito um conjunto de pacotes com o mesmo protocolo de nível 3 e os mesmos endereços de origem e destino. Se o protocolo de nível 3 for IPv4 ou IPv6 é ainda tido em conta o protocolo de nível 4, e se este for TCP ou UDP, as portas de origem e destino. A sintaxe para a instalação desta disciplina de serviço é a seguinte:

```
sfq [perturb SECS] [quantum BYTES]
```

A opção `perturb` permite introduzir periodicamente uma perturbação no cálculo do valor de *hash* por forma a maximizar a equidade na distribuição do débito, evitando colisões repetidas de pacotes dos mesmos fluxos. Esta opção aceita como parâmetro o período, em segundos, em que deve ser alterado o valor da perturbação. A opção `quantum` permite especificar a quantidade de dados, em octetos, que pode ser enviada de cada fila em cada passagem do *round-robin*. Se omitida assume-se igual à MTU na interface, e deve ser sempre maior ou igual a este valor.

A presente implementação do SFQ, por razões de eficiência, tem algumas limitações. O divisor de *hash* é 1024[25], o número de filas é 128 e cada fila pode armazenar até 128 pacotes. Estes limites podem ser alterados, mas apenas no próprio código-fonte antes da compilação.

[24] Por usar tabelas de *hash* não é completamente equitativa, pois "fluxos" em que o valor de *hash* colida são considerados como um só.

[25] Portanto a tabela de *hash* tem 1024 entradas. As colisões de valores de *hash* começam a ser frequentes quando o número de fluxos simultaneamente activos excede 1/10 deste valor.

red

A disciplina de serviço RED é a implementação do algoritmo *Random Early Detection* apresentado na secção 3.8. Para a sua instalação com a ferramenta `tc` usa-se a seguinte sintaxe:

```
red limit BYTES min BYTES max BYTES avpkt BYTES burst PACKETS
[probability PROBABILITY] [bandwidth KBPS] [ecn]
```

O parâmetro `limit` permite especificar o limite físico para o tamanho da fila em octetos. O parâmetro `min` especifica o limiar inferior a partir do qual a probabilidade de eliminação começa a ser não nula. O parâmetro `max` especifica o limiar superior, e deve ser inferior a `limit`, com uma diferença que permita absorver picos de tráfego[26]. Imediatamente acima deste limiar a probabilidade de perda é 1, e imediatamente abaixo deste limiar a probabilidade de perda é a especificada na opção `probability` que, se omitida, assume o valor-padrão 0.02. Note-se que internamente este valor é armazenado como um logaritmo de base 2 e com uma precisão limitada, pelo que é possível que valores configurados ligeiramente diferentes conduzam a valores internos iguais. O parâmetro `avpkt` especifica o tamanho médio esperado para os pacotes, e o parâmetro `burst` especifica o número de pacotes desse tamanho que deve ser possível enviar sem perdas. Estes dois parâmetros (juntamente com o parâmetro `min`) são utilizados para, internamente, parametrizar a função EWMA usada no cálculo do comprimento médio da fila. A opção `bandwidth` que, se omitida, assume o valor de `10Mbit`[27] , indica o débito da interface, e é usada para pré-calcular uma tabela interna usada por uma questão de desempenho.

Uma última opção da disciplina de serviço RED é a `ecn`. Se esta for especificada, em vez de eliminados, os pacotes são marcados com informação de congestionamento, usando o suporte para Notificação Explícita de Congestionamento (ECN) [19] oferecido pelo núcleo do Linux.

gred

A disciplina de serviço `gred` foi introduzida pelo pacote de serviços diferenciados (*linux-diffserv*) para permitir a implementação de múltiplas prioridades de perda, conforme o requerido pelo PHB *Assured Forwarding*. A selecção da fila virtual aplicável ao pacote é feita com base nos 4 *bits* menos significativos do campo `skb->tc_index` do seu cabeçalho interno, inicializado pela pseudo-disciplina de serviço `dsmark`. A instalação da disciplina `gred` é feita em duas fases: na primeira fase é feita uma inicialização geral; na segunda fase são configurados os parâmetros de cada uma das curvas RED. A sintaxe para efectuar a primeira fase da instalação usando a ferramenta `tc` é a seguinte:

```
gred setup DPs NUM_DP default DEF_DP [grio]
```

[26] Note-se que `limit` é um limite físico ao tamanho instantâneo da fila, ao passo que `max` é um limiar para o seu tamanho médio.

[27] Neste contexto, 10×1024^2 bits/segundo, devido à forma como `Mbit` é interpretado pela ferramenta `tc`. Provavelmente o desejado seriam os 10 Mbits/segundo da *Ethernet*.

A palavra `setup` indica que esta é a primeira fase da instalação. O parâmetro `DPs` permite indicar o número de diferentes curvas RED, estando presentemente limitado a um máximo de 16. O parâmetro `default` indica qual a fila em que devem ser colocados os pacotes em que os 4 *bits* menos significativos de `skb->tc_index` seleccionem uma fila inválida ou, se for usado o esquema `grio`, a prioridade estiver mal configurada. Em princípio, esta será a fila com mais elevada probabilidade de perda. A opção `grio` indica que deve ser usado um esquema de partilha de *buffers* em que ao comprimento médio de cada fila é adicionado o comprimento médio de todas as filas de maior prioridade (valor `prio` inferior) para a comparação com os limiares definidos. Assim, se a parametrização de todas as curvas for idêntica, as de maior prioridade têm menores perdas porque vêem sempre a fila mais curta.

Para configurar cada uma das filas virtuais (segunda fase) usa-se o comando `tc class change` com a seguinte sintaxe:

```
gred DP NDP limit BYTES min BYTES max BYTES avpkt BYTES burst PACKETS
[probability PROBABILITY] [bandwidth KBPS] [prio VALUE]
```

O parâmetro `DP` identifica a fila virtual que se pretende configurar. O parâmetro `limit` impõe um limite absoluto ao volume de dados nessa fila. Os parâmetros `min` e `max` definem, respectivamente, os limiares inferior e superior na curva RED. Os parâmetros `burst` e `avpkt` têm o mesmo significado e são usados para o mesmo fim que na disciplina `red`. As opções `probability` e `bandwidth` também têm o mesmo significado que na disciplina `red`, e se omitidas assumem os mesmos valores-padrão. Se a opção `grio` tiver sido especificada na primeira fase da instalação, deve usar-se a opção `prio` para especificar a prioridade da fila virtual.

dsmark

A pseudo-disciplina de serviço `dsmark` é usada, normalmente em conjunto com o filtro `tcindex`, para gerir, num ambiente de serviços diferenciados, a classificação de pacotes com base no campo DSCP, e também para efectuar a (re)marcação dos pacotes à saída. O seu modo de funcionamento é o que a seguir se descreve.

Quando um pacote é passado a uma instância de `dsmark`, é analisado o valor do seu octeto DS, sendo este octeto armazenado no campo `skb->tc_index` do cabeçalho interno do pacote se `dsmark` tiver sido instalado com a opção `set_tc_index`. Em seguida, é invocada a função de classificação que, por sua vez, percorre os filtros que estejam instalados nesta instância de `dsmark`. Se o pacote for seleccionado por algum dos filtros, os 16 *bits* menos significativos do valor resultante da classificação são armazenados, novamente em `skb->tc_index`. Note-se que o resultado da classificação pode ser um valor derivado do octeto DS (gerado por uma instância do filtro for `tcindex`), um identificador de classe, ou qualquer outro valor que um filtro possa retornar.

Se o pacote não for seleccionado ou não existirem filtros pode acontecer uma de duas coisas: se `dsmark` estiver configurado com a opção `default_index`, o seu valor é armazenado em

`skb->tc_index`; caso contrário, este campo é deixado inalterado. Assim, se não for usada nenhuma das opções `default_index` ou `set_tc_index` e nenhum filtro seleccionar o pacote, o campo `skb->tc_index` fica com um valor indefinido. Por outro lado, se apenas for especificada `default_index`, quando é invocada a função de classificação o referido campo ainda tem um valor indefinido, pelo que não pode usar-se o filtro `tcindex` nesta instância de `dsmark` (embora possa ser usado numa sua descendente). À saída é feita a remarcação do octeto DS, conforme adiante se descreve.

A sintaxe para a instalação da disciplina de serviço `dsmark` é a seguinte:

```
dsmark indices INDICES [default_index DEFAULT_INDEX] [set_tc_index]
```

O parâmetro `indices`, que deve ser uma potência inteira de 2, indica o número de classes que vão existir nesta instância da disciplina `dsmark`. Estas classes são automaticamente criadas[28], com a parte inferior do identificador a partir de :1. A opção `default_index`, com um valor entre 0 e 0xffff[29], permite definir um índice que é atribuído ao pacote se a função de classificação não lhe puder atribuir nenhum. A opção `set_tc_index` indica que ainda antes de invocar a função de classificação deve ser colocado em `skb->tc_index` o valor do octeto DS do cabeçalho IP do pacote.

Para configurar as classes criadas usa-se em cada uma o comando `tc class change` com a seguinte sintaxe:

```
dsmark [mask MASK] [value VALUE]
```

Os parâmetros `mask` e `value` controlam a remarcação à saída da disciplina `dsmark`. À saída, os $N = \log_2(\text{INDICES})$ *bits* menos significativos do campo `skb->tc_index` são usados para indexar a tabela de classes, obtendo assim os valores de `mask` e `value` correspondentes. O octeto DS à saída, $\texttt{DS}_o$, é, então, obtido a partir do seu valor à entrada, $\texttt{DS}_i$, segundo a fórmula

$$\texttt{DS}_o = \texttt{DS}_i \;\&\; \texttt{MASK} \mid \texttt{VALUE}$$

Os valores padrão de `MASK` e `VALUE` são, respectivamente, 0xff e 0, pelo que nas classes que não forem explicitamente configuradas não é efectuada remarcação.

É de salientar que as classes de `dsmark` não são realmente classes no sentido de se poderem instalar novas disciplinas de serviço como suas descendentes. Neste caso, as "classes" apenas servem para guardar informação relativa à remarcação. Assim, `dsmark` permite a instalação de uma única disciplina de serviço sua descendente, e que deve usar o *handle* da disciplina `dsmark` no parâmetro `parent`. Uma vez que as classes de `dsmark` não são reais, é possível que um filtro seleccione uma classe que não exista sem que isso constitua qualquer problema: neste caso, a parte inferior do identificador de classe é armazenada em `skb->tc_index` como qualquer outro valor.

[28] Apesar de serem automaticamente criadas, estas classes não são inicializadas. Assim, uma listagem com `tc class ls dev ITF` feita logo após a instalação da disciplina `dsmark` é vazia, mas se se tentar adicionar a classe com `tc class add` o sistema responde que a classe já existe.

[29] Nas versões do núcloe Linux e da ferramenta `tc` utilizados, devido a um erro de concepção, `default_index` estava limitado a valores entre 1 e o número de índices especificado menos 1. Em versões mais recentes este problema foi corrigido.

ingress

A pseudo-disciplina de serviço `ingress` serve para, quando instalada na hierarquia especial de igual designação, permitir a instalação de filtros. A instalação de filtros no ingresso possibilita o policiamento à entrada de fluxos de pacotes por eles seleccionados, de forma semelhante ao que acontece no CAR (*Commited Access Rate*) [20] da Cisco Systems. Tal como `dsmark`, também `ingress` guarda em `skb->tc_index` o resultado da classificação.

A instalação desta pseudo-disciplina de serviço efectua-se correndo simplesmente `tc qdisc add dev ITF ingress`, sendo-lhe automaticamente atribuído o identificador ffff:0.

4.2.8 Filtros

u32

O filtro `u32` permite efectuar a classificação com base em qualquer campo do pacote. As grandes vantagens deste filtro são o seu carácter extremamente genérico e a relativa eficiência de implementação, principalmente com elevados números de chaves. A sua maior desvantagem é uma acrescida complexidade para conseguir uma configuração eficiente.

Sendo um filtro específico, existe um elemento (*knode* terminal) por cada fluxo de pacotes[30] distinto. Para minimizar a degradação de desempenho inerente a um número potencialmente elevado de elementos específicos, o filtro `u32` está estruturado em vários níveis de tabelas de chaves (*knodes*) e de *hash* (*hnodes*). Cada entrada na tabela de *hash* de um *hnode* pode apontar para uma lista ligada de *knodes*, que podem ser terminais ou intermédios, referenciando um *hnode* descendente. A entrada na tabela de *hash* é seleccionada com base num qualquer campo do cabeçalho do pacote. Se a lista de *knodes* da entrada seleccionada não estiver vazia, então esses *knodes* são percorridos por ordem até encontrar um em que todas as chaves sejam satisfeitas pelo pacote. Se se tratar de um *knode* terminal, o pacote é atribuído à classe correspondente. Caso contrário, repete-se o processo com o novo *hnode*, possivelmente efectuando antes algum desencapsulamento. Sempre que chega ao fim de uma lista de *knodes* e o pacote não é seleccionado, o filtro prossegue no *knode* que se segue na lista ao *knode* ancestral directo do *hnode* ao qual pertence a lista.

A raiz desta hierarquia é um *hnode* com apenas uma entrada (automaticamente seleccionada) na tabela de *hash*, existente em cada instância do filtro[31]. Esta pode ser criada instalando `u32` sem qualquer parâmetro e sem especificar o `handle`, mas isto é desnecessário, uma vez que é feito automaticamente quando se instala o primeiro *hnode* ou *knode*.

Os *handles* (identificadores) dos elementos do filtro têm a estrutura XXX:YY:ZZZ, correspondendo ao inteiro de 32 *bits* 0xXXXYYZZZ. Nos *hnodes* a parte XXX é o seu identificador, e as partes YY e ZZZ não têm significado, sendo, portanto, nulas. Nos *knodes* XXX indentifica o *hnode*

[30] Neste contexto, entenda-se fluxo de pacotes como um conjunto de pacotes que têm em comum uma determinada característica.

[31] Neste contexto, entenda-se por instância do filtro o conjunto de todos os elementos desse filtro instalados num dado nó da hierarquia de disciplinas de serviço e classes.

do qual descendem, YY o valor de *hash* que selecciona a lista ligada de *knodes* a que pertence, e ZZZ a sua ordem nessa lista.

Os *knodes* devem ser criados depois do *hnode* seu ancestral directo. Os *knodes* intermédios devem ainda ser criados depois do *hnode* seu descendente directo.

A figura 4.2 mostra um exemplo de hierarquia de hnodes e knodes num filtro u32, correspondente a uma configuração apresentada no anexo C.

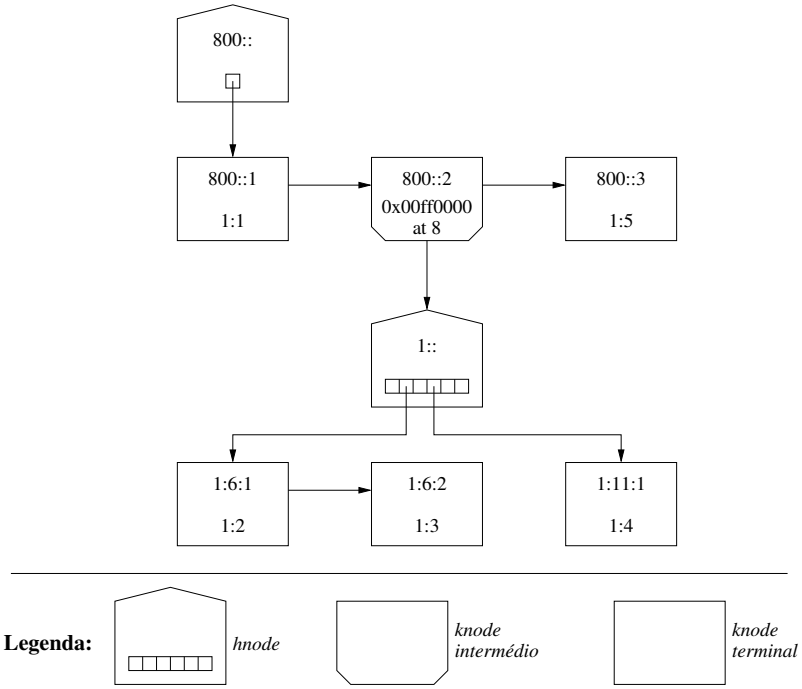

Figura 4.2: Exemplo de hierarquia de elementos num filtro `u32`

Para instalar um *hnode*, a sintaxe é simplesmente

```
u32 divisor DIVISOR
```

onde `DIVISOR` é o número de entradas na tabela de *hash*, entre 1 e 256. Se for especificado o `handle` no comando `tc filter` para identificar o *hnode*, este deve apenas especificar a parte XXX::. Se for omitido é usado um valor atribuído automaticamente a partir de 800::.

Para instalar um *knode*, a sintaxe é a seguinte:

```
u32 [match SELECTOR ...] [offset OFFSET_SPEC] [ht HTID]
[hashkey HASHKEY_SPEC] [order ORDER] [sample SAMPLE]
[classid CLASSID] [link HTID] [police POLICE_SPEC]
```

O identificador `match` permite instalar uma ou mais selectores (chaves) para comparação com os respectivos campos no pacote. Os selectores podem ser de dois tipos: genéricos ou específicos. Sempre que possível é preverível usar selectores específicos, pois têm um significado mais explícito.

Os selectores genéricos, no entanto, permitem analisar qualquer campo de qualquer cabeçalho, incluindo cabeçalhos de protocolos novos ou proprietários. De facto, todos os selectores específicos são internamente convertidos pela ferramenta `tc` em selectores genéricos, o que pode confirmar-se fazendo uma listagem dos filtros.

Na versão utilizada da ferramenta `tc` (2.2.4-now-ss001007), existem selectores específicos para os protocolos IP, IPv6, UDP, TCP e ICMP, identificados respectivamente pelos identificadores `ip`, `ip6`, `udp`, `tcp` e `icmp`. Dentro de cada um destes protocolos é possível analisar diversos campos, conforme a seguir se descreve:

`ip`

`src PREFIX[/LEN]` primeiros `LEN` *bits* (32 se omitido) do endereço de origem iguais aos de `PREFIX`

`dst PREFIX[/LEN]` primeiros `LEN` *bits* (32 se omitido) do endereço de destino iguais aos de `PREFIX`

`tos VAL8 MASK8` *bits* não mascarados[32] do campo `TOS` do cabeçalho iguais aos de `VAL8`. O identificador `tos` pode também designar-se por `dsfield` ou `precedence`.

`ihl VAL8 MASK8` *bits* não mascarados do campo `IHL` do cabeçalho iguais aos de `VAL8`. Na realidade, dependendo da máscara, selecciona também o campo `VERSION` (versão).

`protocol VAL8 MASK8` *bits* não mascarados do campo `PROTOCOL` (protocolo) do cabeçalho iguais aos de `VAL8`.

`nofrag` *flag* indicativa de pacote não fragmentado activa

`firstfrag` *flag* indicativa de primeiro fragmento activa

`df` *flag* "não fragmentar" activa

`mf` *flag* indicativa da existência de mais fragmentos activa

`sport VAL16 MASK16` *bits* não mascarados da porta de origem do protocolo de nível 4 iguais aos de `VAL16`. Assume que esse protocolo é TCP ou UDP e que o cabeçalho IP não contém opções (cabeçalho com apenas 20 octetos), pelo que só é utilizável nessas circunstâncias.

`dport VAL16 MASK16` *bits* não mascarados da porta de destino do protocolo de nível 4 iguais aos de `VAL16`. Sujeito às mesmas restrições que `sport`.

`icmp_type VAL8 MASK8` *bits* não mascarados do campo TYPE (tipo) do protocolo ICMP iguais aos de `VAL8`. Assume que esse protocolo é ICMP e que o cabeçalho IP não contém opções, pelo que só é utilizável nessas circunstâncias.

[32] Ou seja, com valor 1 em `MASK`

`icmp_code VAL8 [MASK8]` *bits* não mascarados do campo CODE (código) do protocolo ICMP iguais aos de `VAL8`. Sujeito às mesmas restrições que `icmp_type`.

`ip6`

`src PREFIX[/LEN]` primeiros `LEN` *bits* (128 se omitido) do endereço de origem iguais aos de `PREFIX`[33]

`dst PREFIX[/LEN]` primeiros `LEN` *bits* (128 se omitido) do endereço de destino iguais aos de `PREFIX`

`flowlabel VAL32 MASK32` *bits* não mascarados (todos) do campo FLOWLABEL do cabeçalho IPv6 iguais aos de VAL32. Na verdade, selecciona também os campos VERSION e/ou PRIORITY se algum dos 8 *bits* mais significativos da máscara estiver activo.

`priority, protocol, dport, sport, icmp_type` e `icmp_code` funcionam de forma semelhante ao que acontece em `ip`.

`udp` ou `tcp`

`src VAL16 MASK16` *bits* não mascarados da porta de origem iguais aos de `VAL16`. Não está sujeito, no entanto, às mesmas restrições que `ip sport`, mas para funcionar correctamente necessita que o parâmetro `offset` esteja bem configurado num *knode* seu ancestral.

`dst VAL16 MASK16` *bits* não mascarados da porta de destino iguais aos de `VAL16`. Sujeito às mesmas condições do anterior.

`icmp`

`type VAL8 MASK8` *bits* não mascarados do campo TYPE iguais aos de VAL8. Sujeito às mesmas condições de `udp src`.

`code VAL8 MASK8` *bits* não mascarados do campo CODE iguais aos de VAL8. Sujeito às mesmas condições do anterior.

Os selectores genéricos podem ser de três tipos: `u8`, para campos de 8 *bits*, `u16` para campos de 16 *bits*, e `u32` para campos de 32 *bits*. Mais uma vez, internamente todos eles são convertidos em selectores `u32`. A sua sintaxe é

```
u(8 | 16 | 32) VAL MASK [at [nexthdr+]OFFSET]
```

devendo `VAL` (o valor a comparar) e `MASK` (a máscara) respeitar o número de bits do selector. O parâmetro `at` permite especificar o deslocamento, em octetos, do campo que se pretende seleccionar em relação ao início do cabeçalho. Esse `OFFSET` deve estar alinhado em fronteiras do mesmo número

[33] Os selectores `src` e `dst` em `ip` ou `ip6` podem ainda ter uma cláusula `at` como os selectores genéricos para indicar uma posição alternativa do campo no cabeçalho, a utilizar em vez da que seria normal.

de *bits* que o próprio selector, isto é, `u8` pode ter qualquer `OFFSET`, `u16` tem que ter `OFFSET` par, e `u32` `OFFSET` múltiplo de 4. Se for usada a opção `nexthdr+`, o deslocamento refere-se não ao início do cabeçalho de nível 3, mas sim ao de nível 4 (ou, se for usado encapsulamento, ao do pacote encapsulado). Para poder funcionar, no entanto, esta opção depende da correcta configuração do parâmetro `offset` num *knode* seu ancestral, uma vez que se isto não acontecer o filtro não consegue determinar a posição onde se inicia o cabeçalho desejado.

Para configurar mais que uma chave no *knode*, é possível especificar mais que uma chave num `match`. Em alternativa, é também possível usar `match` mais que uma vez.

O parâmetro `offset` permite indicar onde se inicia o cabeçalho do protocolo de nível 4 ou do pacote encapsulado. Para tal, dispõe das seguintes opções:

`at OFFSET` indica o deslocamento em relação ao início do cabeçalho do campo (de 16 *bits*) contendo informação sobre o início do cabeçalho seguinte. Este deslocamento deve ser um número par de octetos (alinhamento a 16 *bits*).

`mask MASK` indica a máscara (de 16 *bits*) a aplicar ao campo seleccionado em `at`.

`shift SH` indica o deslocamento[34] de *bits* para a direita a aplicar ao valor do campo depois de aplicada a máscara para obter o valor final da parte variável do deslocamento.

`plus VAL` indica a componente fixa do deslocamento.

`eat` indica que se trata de um pacote encapsulado. Assim, todos os selectores em *knodes* descendentes passam a ter como referência o início do cabeçalho interior, e não apenas aqueles em que é usada a opção `nexthdr+` no parâmetro `at`.

Sobre a configuração de `offset` há ainda dois aspectos que convém referir. Um é que o filtro assume que o início de um cabeçalho de nível superior ou um cabeçalho encapsulado estão sempre alinhados a 32 *bits* (4 octetos). Além disso, assume também que o campo de onde se obtém o deslocamento não cruza uma fronteira de 16 *bits*, e `shift` não permite deslocamentos para a esquerda. No caso do IP isto não põe problemas, dada a localização do campo IHL, mas com outros protocolos pode não ser assim. O segundo aspecto é que, provavelmente devido a um *bug* no filtro, não é possível usar apenas `plus` para assim conseguir um deslocamento fixo[35]. Isto pode contornar-se indicando também uma máscara com todos os *bits* a zero, que implica um deslocamento variável sempre nulo, que será então somado ao fixo.

O parâmetro `ht` permite indicar qual o *hnode* de que descende o *knode* a ser configurado. Se omitido, é usado 800::, o *hnode* raiz.

[34] Operação de deslocamento (*shift*). Salvo indicação em contrário, deslocamento é usado com significado de distância entre o início de um campo e o início do cabeçalho a que esse campo pertence, i.e., *offset*.

[35] Note-se que isto só é problemático no caso de encapsulamento (`eat`), uma vez que nos restantes casos pode sempre adicionar-se esse deslocamento directamente ao parâmetro `at` no *knode* descendente, evitando a especificação como `offset`.

Em *knodes* intermédios, o identificador `hashkey` permite indicar o campo que serve de chave para a tabela de *hash* do *hnode* seu descendente. Esse campo pode ter até 32 *bits*, sendo usada uma máscara para seleccionar apenas o que interessa. A sintaxe para `hashkey` é

```
hashkey [mask MASK] [at OFFSET]
```

em que `mask` permite especificar a máscara de 32 bits e `at` o deslocamento (múltiplo de 4 octetos) em relação ao início do cabeçalho. O valor de *hash* resultante é igual ao XOR dos 4 octetos abrangidos depois de mascarados.

O parâmetro `order` permite indicar em que posição o *knode* vai ficar na lista, caso existam outros na mesma entrada da tabela de *hash* do *hnode* do qual descende. Especifica, portanto, a parte ::ZZZ do *handle*. Os *knodes* de uma lista são testados por ordem crescente de ::ZZZ. Se `order` for omitido, é automaticamente atribuído um valor a partir de 2048, correspondente a ::800 no *handle*[36].

O identificador `sample` fornece ao utilizador uma maneira fácil de instalar um *knode* como descendente de uma dada entrada da tabela de *hash* de um *hnode*. Usando a mesma sintaxe que `match` (com a diferença de que admite apenas um selector) o utilizador indica ao sistema o índice da entrada na tabela a que corresponde. Por exemplo, se no *knode* seu antecessor mais próximo `hashkey` tiver sido configurado para seleccionar o IP de origem (`hashkey mask 0xffffffff at 12`), o utilizador deve usar `sample ip src PREFIX/32` para instalar o *knode* na lista correspondente ao valor de *hash* calculado a partir de `PREFIX`. Em alternativa, o utilizador poderia especificar a parte :YY: no parâmetro `ht`, mas para isso teria que calcular ele próprio o valor de *hash* correspondente.

Note-se que os valores de *hash* têm apenas 8 *bits*. Assim, se o campo usado para o cálculo do valor de *hash* tiver mais que 8 *bits*, podem ocorrer colisões, tornando necessário repetir a cláusula `sample` numa `match` para distinguir entre os casos que colidem, se isso for necessário. Se a tabela de *hash* tiver menos que 256 entradas, apenas são usados os $\log_2 N$ *bits* menos significativos[37] do valor de *hash* para seleccionar a entrada (*bucket*). Isto pode também dar origem a colisões de *hash*, pelo que pode também obrigar ao uso de cláusulas `match` para distinguir casos que colidam.

Num *knode* terminal o parâmetro `classid` (que também pode designar-se por `flowid`) indica qual a classe à qual devem ser atribuídos os pacotes seleccionados. Num *knode* intermédio, pelo contrário, deve usar-se o parâmetro `link` para identificar o *hnode* seu descendente.

A opção `police` permite instalar um elemento de policiamento para os pacotes seleccionados.

Como se pode concluir do que foi referido, as grandes vantagens do filtro `u32` são o seu carácter extremamente genérico e a relativa eficiência de implementação, principalmente com elevados números de chaves. A sua maior desvantagem é uma acrescida complexidade para conseguir uma configuração eficiente.

[36] Isto porque o parâmetro `order` aceita um número inteiro, que se assume decimal a menos que inclua o prefixo 0x, enquanto o *handle* é sempre representado em notação hexadecimal.

[37] É efectuada uma operação binária AND do valor de *hash* com `divisor - 1`

Por fim, convém referir que devido a um erro (*bug*) na implementação deste filtro não é possível repetir `handle`s em filtros instalados no mesmo `parent`, ainda que tenham valores diferentes de `pref`.

fw

Este filtro, originalmente concebido como filtro genérico, pode, nas versões mais recentes do núcleo Linux, configurar-se também como filtro específico. Ele permite usar a marcação efectuada pela camada de *firewall* para classifição de pacotes.

Para a instalação como filtro genérico não é usado qualquer parâmetro, devendo ainda omitir-se o parâmetro `handle` no comando `tc filter` usado para efectuar a instalação. Neste caso, para ser possível a atribuição directa do pacote a uma classe, a marca deve ser um número de 32 *bits* em que os 16 mais significativos contêm a parte superior e os 16 menos significativos a parte inferior do identificador da classe.

Para instalar o filtro `fw` como específico usa-se a seguinte sintaxe, tantas vezes quantos os elementos que se prentende instalar:

```
fw classid CLASSID [police POLICE_SPEC]
```

O parâmetro `classid` indica a classe a que devem ser atribuídos os pacotes cuja marcação efectuada pela camada de *firewall* (`skb->nfmark`) contenha o valor especificado no parâmetro `handle` do comando `tc filter` usado para efectuar instalação, que é obrigatório neste caso. O parâmetro `police` permite instalar policiamento para os pacotes seleccionados.

É de salientar que mesmo que o filtro `fw` estivesse previamente instalado como genérico, assim que se instala um elemento específico ele passa a comportar-se como filtro específico, ainda que o elemento instalado seja removido.

Note-se que é possível instalar um elemento específico usando em `classid` um identificador de uma classe que ainda não foi criada. No entanto, uma vez que não é possível obter o identificador interno da classe, o elemento não pode armazená-lo, o que pode levar a que os pacotes necessitem de ser reclassificados. Por outro lado, é também possível usar um elemento específico com um identificador de classe em que a parte superior é nula (portanto inválido), o que permite usar a forma específica do filtro `fw` em casos como o nó `split` da disciplina de serviço CBQ (ver secção 3.6).

route

O filtro `route` permite usar marcações efectuadas pela camada de encaminhamento (*routing*) do sistema operativo para classificar os pacotes. Originalmente concebido como filtro genérico, `route` pode também ser usado como filtro específico. De facto, devido a alterações no código de encaminhanento de versões recentes do núcleo Linux, a sua utilização como filtro genérico está bastante

limitada[38], pelo que tenderá a desaparecer.

Tal como no caso anterior, para instalar o filtro `route` como genérico não se usa qualquer parâmetro. Para a instalação como filtro específico usa-se a seguinte seguinte sintaxe para instalar cada um dos elementos:

```
route [from REALM | fromif TAG] [to REALM] [order ORDER]
flowid CLASSID [police POLICE_SPEC]
```

A opção `from` permite especificar a região (*realm*) de origem do pacote, e é mutuamente exclusiva com a opção `fromif`, que permite indicar a interface de entrada do pacote. A opção `to` permite especificar a região de destino do pacote. A opção `order` permite especificar uma ordem adicional, entre 0 (valor padrão) e 127, para avaliação de elementos do mesmo tipo. O parâmetro `flowid` (que também pode designar-se `classid`) permite especificar o identificador da classe à qual se pretendem atribuir os pacotes seleccionados. A opção `police` permite instalar e configurar policiamento para o tráfego seleccionado.

Quer se instale como filtro genérico ou específico, deve omitir-se o parâmetro `handle` do comando `tc filter` usado para o instalar[39]. De facto, o `handle` é obtido a partir da combinação dos parâmetros `from` ou `fromif`, `order` e `to`.

A ordem de avaliação dos elementos é a mesma da listagem obtida com `tc filter ls`: primeiro todos os elementos em que são especificadas tanto a região `to` como a região `from`; em seguida, todos os elementos em que apenas foi especificada a região `to`; por último aqueles em que apenas foi especificada a região `from`. Dentro de cada um destes conjuntos os elementos são avaliados por ordem crescente de `order`. Isto pode ser útil por exemplo para atribuir os pacotes conformes de um fluxo a uma classe e os não-conformes, marcados para reclassificação pelo módulo de policiamento, a outra classe.

rsvp

O filtro `rsvp` permite classificar os pacotes com base nos parâmetros que definem o fluxo RSVP (ver secção 2.5.4) ao qual pertence. A sintaxe para a instalação deste filtro é a seguinte:

```
rsvp ipproto PROTOCOL session DST[/PORT | GPI] [sender SRC[/PORT | GPI]
[classid CLASSID] [police POLICE_SPEC] [tunnelid ID] [tunnel ID skip NUMBER]
```

onde `GPI` pode ser qualquer um dos seguintes:

```
flowlabel NUMBER
```

[38] Embora o filtro continue a incluir toda a funcionalidade, o código de encaminhamento não consegue marcar o campo `skb->dst->tclassid` com o identificador de qualquer classe: apenas se conseguem seleccionar classes em que tanto a parte superior como a inferior do seu identificador sejam menores ou iguais a 255 (0xFF). Para o fazer, é necessário usar os comandos `ip rule` ou `ip route` com a opção `realms S:I`, em que `S` e `I` são, respectivamente, a parte superior e inferior do identificador de classe, sujeitas à referida restrição.

[39] Se a instalação for como filtro específico pode especificar-se o `handle`, mas tem que coincidir com o calculado.

```
spi/ah SPI
spi/esp SPI
u(8|16|32) NUMBER mask MASK at OFFSET
```

O parâmetro `ipproto` indica o protocolo de nível 4 encapsulado no datagramas IP do fluxo a seleccionar. Os valores possíveis para este parâmetro encontram-se no ficheiro `/etc/protocols`. O parâmetro `session` permite identificar a sessão RSVP através do endereço IP de destino e, opcionalmente, a porta de destino, que pode tomar a forma de porta generalizada — GPI (*Generalized Port Identification*). O parâmetro opcional `sender` (que também pode designar-se `flowspec`) permite fazer a selecção do emissor usando o endereço IP de origem e a porta (generalizada) de origem, podendo esta última omitir-se. O parâmetro `classid` (que também pode designar-se `flowid`) permite seleccionar a classe à qual se devem atribuir os pacotes seleccionados.

O filtro `rsvp` prevê a utilização de túneis, pelo que são disponibilizadas opções para tratar estes casos. Ao especificar o elemento do filtro que selecciona os pacotes do túnel (encapsuladores), em vez de `classid` (ou `flowid`) devem usar-se os parâmetros `tunnel` e `skip`. O primeiro permite atribuir um identificador numérico, entre 1 e 255, ao túnel, enquanto o segundo permite indicar o tamanho do cabeçalho externo, que é usado como deslocamento (*offset*) para o início do cabeçalho encapsulado. No elemento do filtro que faz a selecção dos pacotes encapsulados deve usar-se o parâmetro `tunnelid` com o identificador atribuído ao túnel para indicar que a selecção apenas deve abranger pacotes encapsulados nesse túnel. A selecção de pacotes encapsulados é feita em várias passagens: uma por cada nível de encapsulamento mais uma para os pacotes propriamente ditos.

Por fim, o parâmetro opcional `police` permite instalar policiamento no fluxo dos pacotes seleccionados.

Quanto ao identificador de porta generalizada [21], existem várias especificações alternativas. A opção `flowlabel` permite fazer a selecção com base no campo Flow Label do cabeçalho IPv6. A opção `spi/ah` permite indicar o índice de parâmetros de segurança (SPI) do cabeçalho de autenticação IP (AH) [22]. A opção `spi/esp` permite fazer o mesmo que a anterior, mas para um cabeçalho ESP (IP encapsulating security payload) [23]. Por fim, a alternativa mais genérica é a selecção de um qualquer valor de um dos cabeçalhos. Isso consegue-se usando uma das opções `u8`, `u16` ou `u32`, para seleccionar campos de 8, 16 ou 32 bits, juntamente com uma máscara (*mask*) e um deslocamento (*offset*) em relação ao início do cabeçalho de nível 4[40]. Os pacotes seleccionados são aqueles que no campo com o tamanho e o deslocamento indicados contenham um valor igual à máscara especificada.

Em princípio, qualquer selecção efectuada pelo filtro `rsvp` poderia ser igualmente feita pelo filtro `u32`. No entanto, o primeiro foi especificamente concebido para utilização com RSVP, pelo que, neste caso, é computacionalmente mais eficiente, além de que é mais simples de configurar. Por este motivo, em ambientes RSVP ou outros com requisitos semelhantes deve preferir-se o seu uso ao do filtro `u32`.

[40] Ou, mais propriamente, do cabeçalho mais exterior contido na secção de dados do datagrama IP.

tcindex

O filtro `tcindex` permite usar o campo interno `skb->tc_index` do cabeçalho interno do pacote para efectuar a classificação dos pacotes. Este filtro é, normalmente, usado dentro de uma instância da pseudo-disciplina de serviço `dsmark` (ou de uma sua descendente), que pode inicializar este campo com o valor do octeto DS do cabeçalho IP (e eventualmente modificá-lo).

A sintaxe para a instalação deste filtro é a seguinte:

```
tcindex [hash SIZE] [mask MASK] [shift SHIFT]
[pass_on | fall_through] [classid CLASSID] [police POLICE_SPEC]
```

A opção `hash` permite indicar o tamanho da tabela interna de busca, até um máximo de 0x10000 (65536). Se omitida, é-lhe atribuído um valor padrão obtido a partir do parâmetro `mask` e, em versões mais recentes, também do parâmetro `shift`. Uma vez que está mais relacionada com a optimização da eficiência do que com a funcionalidade propriamente dita do filtro, esta opção pode omitir-se na maioria dos casos. As opções `mask` e `shift` permitem especificar, respectivamente, uma máscara e um deslocamento a aplicar ao valor armazenado em `skb->tcindex`. O valor `(skb->tcindex & MASK) >> SHIFT` pode então ser comparado com os `handle`s dos elementos especficos para efectuar a classificação.

Para os elementos específicos (nos quais é especificado o parâmetro `handle` de `tc filter`) o parâmetro `classid` permite identificar a classe à qual se devem atribuir os pacotes seleccionados. A opção `fall_through` indica que deve ser usado um mapeamento algorítmico se o pacote não for seleccionado. A opção `pass_on`, pelo contrário, indica que se o pacote não for seleccionado deve passar o controlo ao filtro seguinte. Se omitidas, o filtro assume `fall_through`. O mapeamento algorítmico consiste em seleccionar a classe cujo identificador tem a parte superior igual à da disciplina de serviço[41] em que o filtro está instalado e a parte inferior igual a `(skb->tcindex & MASK) >> SHIFT`.

Uma vez que a disciplina de serviço `dsmark` guarda em `skb->tcindex` o resultado da classificação, uma utilização comum do filtro `tcindex` consiste na sua instalação, sem elementos específicos e com `MASK`=0xfc e `SHIFT`=2, numa instância de `dsmark`, por forma a colocar nos bits inferiores de `skb->tc_index` apenas o DSCP em vez de todo o octeto DS.

A opção `police` permite, como de costume, instalar policiamento no fluxo de pacotes seleccionados.

4.3 Geradores/medidores de tráfego

Um componente de grande importância em qualquer sistema de teste de qualidade de serviço é o gerador de fluxos de teste. Este deve ser capaz de gerar fluxos com características tão próximas quanto possível dos gerados pelas aplicações com requisitos de QoS que a rede deve suportar e cujo desempenho se pretende avaliar. Nomeadamente no caso de serviços de tipo CBR, é necessário que a fonte seja capaz de emitir pacotes a uma cadência constante, sem flutuações. Se, pelo contrário,

[41] Ou classe, como pode acontecer na disciplina CBQ com mais que dois níveis.

existirem valores suficientemente elevados de *jitter* de transmissão[42], o fluxo pode receber da rede um serviço inferior ao normal por perder a conformidade com os parâmetros contratados.

Se o gerador de tráfego correr num sistema multitarefa, como é o Linux, é possível que fique inibido de transmitir por certos períodos de tempo durante os quais são executados outros processos. Normalmente, a fatia de tempo no Linux é de 10ms, pelo que se o gerador não correr apenas por uma ou duas fatias de tempo o *jitter* de transmissão ascende a valores da ordem de 10 ou 20ms. Como é referido no apêndice A, é possível reduzir a fatia de tempo para 1ms, e também configurar o mecanismo de escalonamento por forma a que este processo seja corrido com prioridade absoluta sobre todos os outros, conseguindo-se assim valores de *jitter* de transmissão bastante baixos.

Por vezes é necessário gerar mais que um fluxo de teste, nomeadamente para efectuar comparações num ambiente de serviços diferenciados entre fluxos de iguais características mas pertencentes a diferentes agregados de tráfego. A aplicação geradora deve, neste caso, ser capaz de emitir ambos os fluxos sem recurso a *multithreading*, pois a comutação entre *threads* pode conduzir a *jitter* de transmissão, particularmente em sistemas onde o mapeamento entre *threads* e processos é de um para um, como acontece no Linux. Se isto não acontecer, os benefícios do escalonamento prioritário são perdidos.

Quanto à aplicação receptora, ela deve ser capaz de receber os pacotes o mais rapidamente possível após a sua chegada à máquina de destino, para que esse intervalo de tempo não afecte os resultados. Deve ainda permitir a obtenção de estatísticas que permitam avaliar o desempenho da rede, nomeadamente no que se refere a atraso, *jitter*, percentagem de pacotes perdidos e entregues fora de ordem e débito efectivo (*throughput*).

De entre as numerosas aplicações de geração de tráfego e teste de desempenho de redes analisadas, a escolha recaiu sobre o programa `rude`, que consiste num gerador de tráfego (`rude`) e num colector (`crude`) que permite efectuar um registo de parâmetros dos pacotes recebidos. A principal vantagem deste programa é a de cumprir os requisitos de geração precisa de fluxos CBR. Este programa apresentava, no entanto, três limitações: por um lado o colector não gerava automaticamente estatísticas, permitindo apenas registar dados sobre os pacotes recebidos num ficheiro; por outro lado não suporta escalonamento prioritário do lado do colector; por fim, não dispunha de suporte para alteração do campo TOS do cabeçalho IP. O facto de registar num ficheiro implica que, mais cedo ou mais tarde, acabará por ser feito um acesso ao disco, impedindo o programa de correr por um tempo que pode adicionar *jitter*[43] ao fluxo de pacotes recebido. O segundo factor pode também contribuir para aumento do *jitter* medido. Uma vez que o programa é distribuído segundo a licença GPL[44], este foi alterado por forma a colmatar essas lacunas. O *patch* criado para

[42]Neste contexto entenda-se por *jitter* de transmissão o valor absoluto da diferença entre o intervalo de tempo entre a transmissão do pacote i e do pacote i+1 e o intervalo de tempo entre a transmissão do pacote i+1 e do pacote i+2.

[43]Neste texto, a definição de *jitter* está de acordo com a usada em [24], com a diferença de que não é usada a função EWMA, pois neste contexto todos os pacotes são igualmente importantes. Assim, convém distinguir esta definição da normalmente usada no âmbito do ATM, que é o desvio padrão em relação ao instante nominal de transmissão das células.

[44]GNU Public License — Licença de código aberto que permite modificar o código e distribuir a versão modificada sem qualquer restrição excepto que o código modificado deve ser distribuído sob a mesma licença.

alteração do código da versão 0.50 do programa `rude` é apresentado no anexo B.

Para geração do tráfego de enchimento, uma vez que não apresenta requisitos temporais apertados, além da ferramenta `rude`, foram também usadas as aplicações `iperf` e `ttcp`, nomeadamente para a geração e recepção de fluxos TCP. Foi ainda usada a sobejamente conhecida ferramenta de teste de rede `tcpdump`.

4.4 Ambiente de teste

O ambiente de teste utilizado baseia-se em computadores pessoais dispondo de interfaces de rede *fast ethernet* (100baseTX) e ATM (155Mbps). Neste caso a interface de rede ATM é particularmente útil, pois permite a configuração de ligações (*links*) virtuais de baixo débito usando circuitos virtuais com categoria de serviço CBR. Uma ligação de baixo débito é facilmente saturada ainda que as máquinas não sejam muito poderosas, forçando a que o estrangulamento ocorra na ligação em si e não por falta de capacidade de processamento dos nós. Além disso, o facto de permitir ligações ponto-a-ponto em meio não partilhado, torna o comportamento da rede mais determinístico, pois não vai depender de outro tráfego eventualmente existente.

4.4.1 Configuração física

A configuração física baseia-se em três computadores pessoais, ligados a um comutador ATM e a um comutador *ethernet*. O gerador de enchimento serve para gerar tráfego de fundo que vai carregar a rede. No *router* estão implementados os mecanismos de classificação e controlo de tráfego que se pretendem testar. O terceiro computador acumula as funções de gerador e receptor de tráfego. Ele dispõe de dois processadores em configuração de multiprocessamento simétrico, o que lhe permite correr os dois processos sem que interfiram um com outro. As interfaces usadas para emitir e receber o tráfego são também diferentes, evitando o conflito que afectaria as medições. O processo gerador de tráfego é responsável por gerar os fluxos de teste com a maior precisão possível. O processo de recepção, por sua vez, tem por função recolher informação sobre os pacotes recebidos e gerar estatísticas que permitam avaliar o desempenho dos componentes em teste. Existe ainda um outro receptor, para tráfego de enchimento TCP. As razões para a existência desta quarta máquina são por um lado a necessidade de existir um processo para receber este tipo de tráfego (ao contrário do UDP, onde os pacotes podem ser simplesmente eliminados) e por outro lado a dificuldade em correr tal processo no próprio receptor de teste, uma vez que tem os processos de geração e recepção de tráfego de teste a correr com escalonamento prioritário, o que limita as possibilidades de processos adicionais de recepção de tráfego correrem sem problemas. A figura 4.3 mostra a configuração referida.

A razão para usar a mesma máquina para gerar e receber o tráfego de teste prende-se com a necessidade de medir os atrasos introduzidos pela rede. Desta forma os valores de atraso medidos são sempre os reais, não dependendo da sincronização de relógios. Testes previamente realizados mostraram que mesmo usando o protocolo NTP (*network time protocol*) sobre *Fast Ethernet*, a

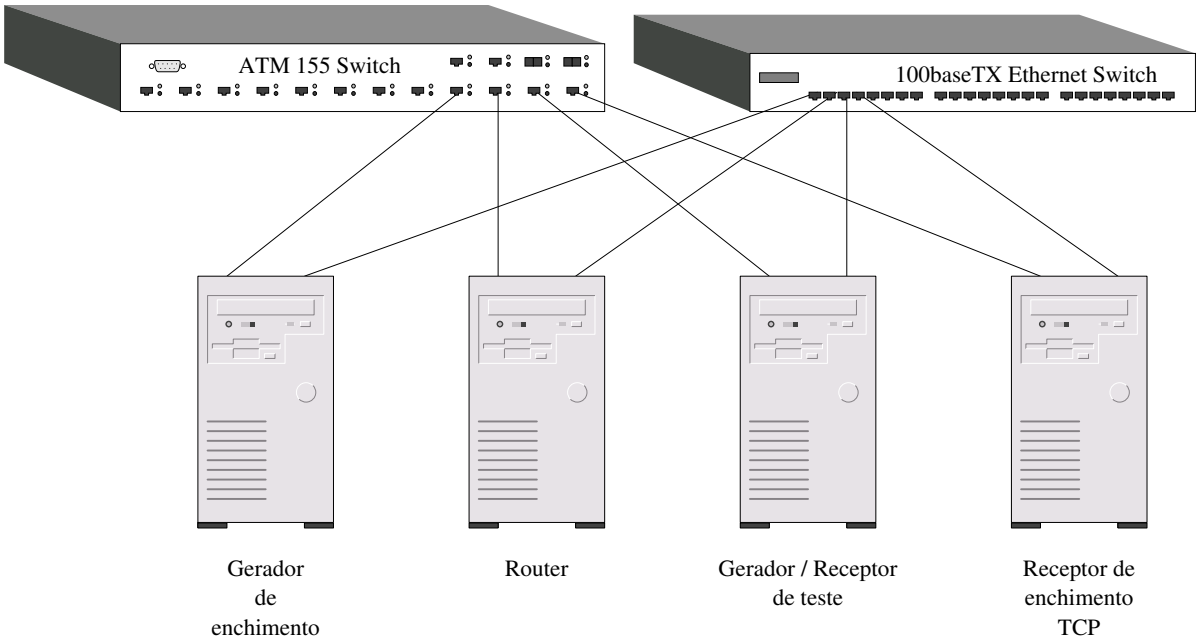

Figura 4.3: Configuração física

margem de tolerância de sincronização entre os relógios nunca descia abaixo de algumas dezenas de milissegundo, ficando normalmente pela ordem de grandeza de alguns décimos de segundo, o que é demasiado elevado quando se pretendem medir atrasos num só *router*[45]. Por outro lado, é interessante obter uma configuração que permita utilizar uma só máquina com duas interfaces, usadas como pontas de prova para inserir em dois pontos distintos da rede a testar, pois constitui uma excelente plataforma para numerosos testes de rede.

4.4.2 Configuração lógica

A configuração lógica usada na maior parte dos testes efectuados é a descrita na figura 4.4, consistindo em duas sub-redes *fast ethernet* comutadas, uma sub-rede IP sobre ATM e uma sub-rede virtual. Na primeira sub-rede *fast ethernet* estão ligados os geradores de fluxos de teste e de enchimento, além do próprio *router*. A sub-rede IP sobre ATM interliga o *router* e a máquina de destino, sendo a ligação estabelecida sobre um PVC com categoria de serviço CBR com um débito relativamente baixo (2Mbps). O objectivo de ter um débito baixo nesta ligação é o de facilmente saturar a sua capacidade por forma a permitir observar o comportamento dos mecanismos de QoS em teste. A outra sub-rede fast ethernet interliga o receptor de fluxos de teste (que aqui actua também como *router*) e o receptor de fluxos TCP de enchimento. Por fim, a sub-rede virtual é obtida configurando uma interface `dummy` no receptor. Este tipo de interface limita-se a eliminar silenciosamente todos os pacotes que lhe sejam passados, permitindo criar uma série de receptores virtuais para o tráfego UDP de enchimento.

[45] Provavelmente isto deve-se à má qualidade dos relógios utilizados nos computadores pessoais.

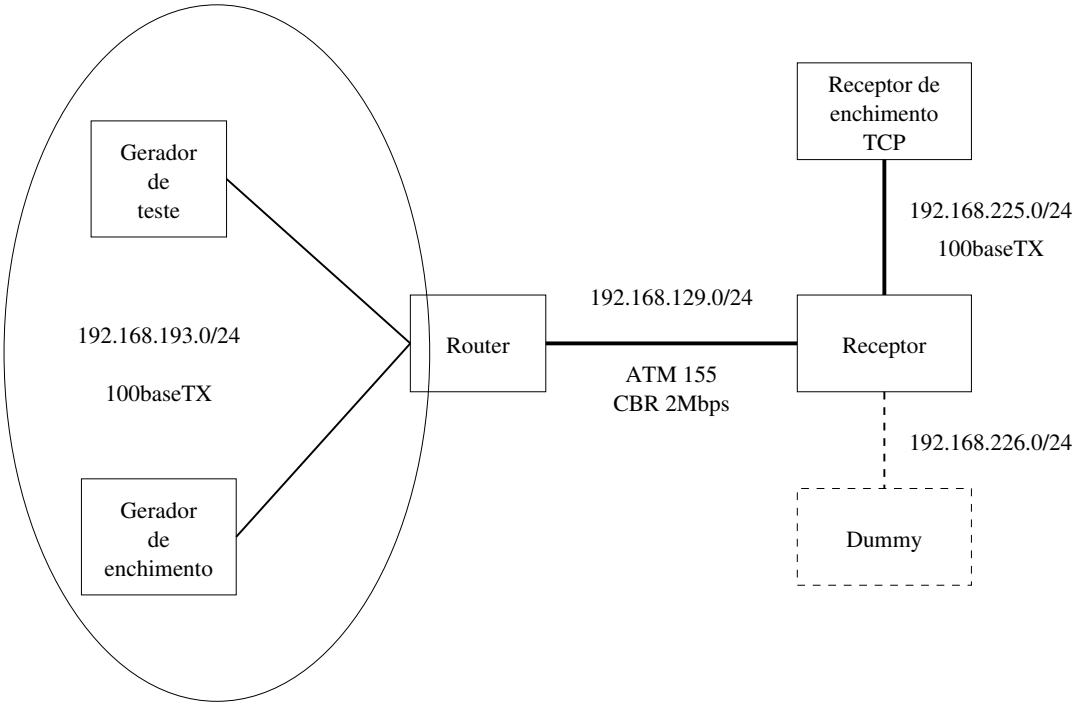

Figura 4.4: Configuração lógica

É de salientar que apesar de o gerador e o receptor de tráfego, por razões de ordem prática, estarem fisicamente na mesma máquina, eles são entidades conceptualmente distintas, razão pela qual aparecem separados no diagrama de configuração lógica.

O facto de ter na mesma máquina os processos de geração e recepção dos fluxos de teste põe dois problemas de ordem prática:

1. A máquina não envia para a rede tráfego destinado a uma das interfaces locais. Este tipo de tráfego é encaminhado internamente, nunca chegando a ser transmitido.

2. Se um pacote recebido tem o endereço IP de origem igual ao de uma das interfaces locais, a pilha TCP/IP elimina-o como medida de segurança, por supor tratar-se de um pacote com esse endereço falsificado.

Para contornar estes problemas é necessário recorrer a técnicas de tradução de endereços de rede (*network address translation* — NAT). A técnica usada necessita de dois endereços IP em cada interface, um real e um fictício. Designando as interfaces por `ITF0` e `ITF1`, e os respectivos endereços real e fictício por `RealIP0`, `FakeIP0`, `RealIP1` e `FakeIP1`, o objectivo é levar o processo gerador a pensar que está a comunicar com uma máquina de endereço `FakeIP1` e o processo receptor a pensar que está a comunicar com uma máquina de endereço `FakeIP0`.

O tráfego destinado a `FakeIP1` é encaminhado pela interface `ITF0` e, inversamente, o tráfego

destinado a `FakeIP0` é encaminhado pela interface `ITF1`. À saída pela interface `ITF0`, é feita uma alteração ao endereço de origem, que passa de `RealIP0` para `FakeIP0`, aos pacotes destinados a `FakeIP1`. Ao receber o pacote na interface `ITF1` é efectuada a alteração do endereço de destino de `FakeIP1` para `RealIP1`. Desta forma a pilha TCP/IP passa o pacote, que parece vir de `FakeIP0`, ao processo receptor. Em sentido contrário (apenas aplicável a fluxos TCP) é feita uma operação simétrica. Assim, à saída pela interface ITF1, o tráfego destinado a `FakeIP0` vê o seu endereço de origem alterado de `RealIP1` para `FakeIP1`. À chegada, na interface `ITF0`, o seu endereço de destino é alterado de `FakeIP0` para `RealIP0` por forma a que a pilha TCP/IP o encaminhe para processamento local. Por fim, para permitir a recepção de tráfego destinado aos endereços fictícios, é necessária uma de duas coisas:

1. Instalar *proxy* ARP para responder na interface `ITF0` a pedidos de resolução do endereço `FakeIP0` com o endereço MAC dessa interface, e na interface `ITF1` a pedidos de resolução do endereço `FakeIP1` com o endereço MAC desta;

   ou

2. Usar como endereços fictícios endereços de outras sub-redes para as quais o tráfego seja encaminhado pelos respectivos endereços reais pelas máquinas externas.

Neste caso, na interface 192.168.193.96, *fast ethernet*, é usada a primeira opção, e na interface 192.168.129.96, IP sobre ATM, é usada a segunda por não existir o conceito de *proxy* ARP neste caso. Além disso, o facto de o endereço fictício usado pertencer à mesma sub-rede do receptor de fluxos de enchimento TCP (192.168.225.0/24) torna a configuração mais simples e coerente.

O *script* genérico para correr no gerador/receptor de teste é apresentado no anexo D.

Capítulo 5

Configuração e teste

Neste capítulo são abordados a configuração de serviços e os testes efectuados para avaliação do seu desempenho. Antes, porém, será feita uma série de testes às próprias ferramentas usadas para que se possa avaliar em que medida são adequadas para a obtenção de resultados significativos e qual a precisão que delas se pode esperar.

5.1 Ferramenta de geração de tráfego: rude

Antes de efectuar qualquer teste é necessário saber se as ferramentas usadas para o fazer são adequadas, permitindo a obtenção de resultados válidos. Os primeiros testes efectuados visam, pois, avaliar as próprias ferramentas de teste para validar o seu uso e conhecer as suas limitações.

Para avaliar a qualidade dos fluxos gerados foi efectuado um teste baseado na geração de um fluxo de débito constante de pacotes UDP. Os pacotes do fluxo têm um tamanho fixo de 64 octetos e são emitidos a uma cadência de 500 por segundo, correspondendo a um débito de 32000 octetos por segundo, ou 256 kbps. A duração dos testes é de 60 segundos[1].

Foi efectuado um segundo teste baseado na geração de dois fluxos idênticos ao anterior. Este é o cenário mais útil em testes de diferenciação de serviços, pois permite comparar directamente o comportamento da rede face a dois fluxos com características idênticas mas aos quais é oferecido um serviço diferente. O objectivo deste segundo teste é essencialmente observar em que medida a geração simultânea de um segundo fluxo afecta a geração de fluxos com boas características temporais.

Por fim foi efectuado um terceiro teste em que o débito do segundo fluxo foi alterado de 500 para 570 pacotes por segundo. Pretende-se assim observar em que medida isto afecta a qualidade dos fluxos gerados. Interessa, nomeadamente, saber em que medida o facto de haver momentos em que o instante nominal de transmissão de pacotes dos diferentes fluxos é quase simultâneo influencia o *jitter* de transmissão[2].

[1] Duração de todos os testes neste capítulo, salvo quando expressamente indicado outro valor.
[2] Componente do *jitter* associada ao processo de geração.

Este conjunto de testes foi efectuado duas vezes: uma usando o escalonamento normal e outra recorrendo ao mecanismo de escalonamento prioritário descrito no apêndice A.

Uma vez que este conjunto de testes não depende do atraso sofrido pelos pacotes, foi usado o método de armazenamento de informação sobre os pacotes num ficheiro para posterior análise. Para cada ficheiro gerado foram calculados os valores médio e máximo do *jitter* de transmissão e o número de intervalos entre pacotes utilizados no cálculo do jitter de cada um dos fluxos. Foram enviados 30000 pacotes no fluxo 1, correspondendo a 29999 intervalos. Destes, os dois primeiros foram eliminados, portanto a existência de 29997 intervalos indica que nenhum pacote foi perdido, o mesmo acontecendo em relação ao fluxo 2.

Os resultados obtidos são apresentados nas tabelas 5.1 e 5.2, sendo os tempos indicados em μs.

Escalonamento	*jitter* médio	*jitter* máximo	Nº de intervalos
normal	1	799	29997
prioritário	1	28	29997

Tabela 5.1: Transmissão de 1 só fluxo

		Fluxo 1			Fluxo 2		
	Escalonamento	*jitter* médio	*jitter* máximo	Nº de intervalos	*jitter* médio	*jitter* máximo	Nº de intervalos
fluxos iguais	normal	1	1605	29997	4	1573	29997
	prioritário	1	125	29997	3	130	29997
fluxos diferentes	normal	2	1166	29997	2	182	34204
	prioritário	2	104	29997	2	113	34204

Tabela 5.2: Transmissão de 2 fluxos

Como pode observar-se, o valor médio do *jitter* de transmissão é sempre muito baixo, na ordem de muito poucos microssegundos. No entanto, e apesar de o núcleo Linux estar compilado com fatias de tempo de aproximadamente 1 milissegundo (`HZ`=1024), o valor de pico do *jitter* de transmissão chega a atingir mais de 1.5 ms[3], mesmo estando a máquina sem qualquer processo adicional a correr, excepto os *daemons* do sistema. Com intervalos nominais entre a transmissão de pacotes de 2 ms, este valor de pico ainda é significativo. Com recurso ao escalonamento prioritário, no entanto, verifica-se que os valores de *jitter* de transmissão são consistentemente muito baixos, ficando o pico sempre bem abaixo de 1 ms (sempre abaixo dos 200 μs neste conjunto de testes).

Quanto à parte da recepção, é de referir que se a ferramenta `crude` for usada sem a opção de escalonamento prioritário existem algumas perdas que, embora sejam pequenas (da ordem de 0.1%), não são negligenciáveis. Fazendo uso dessa opção, no entanto, não se verificam quaisquer perdas. O uso desta opção na recepção não afecta grandemente o sistema, pois sendo as chamadas a `recvfrom()` bloqueantes, é dada a oportunidade aos restantes processos de correr até à chegada do pacote seguinte.

[3]De facto, noutros testes efectuados verificou-se que ocasionalmente atingia valores ainda mais elevados, perto dos 5 ms.

Pior que induzir perdas na própria aplicação que, de alguma forma, falseiam os resultados obtidos, é o grande aumento no valor de pico do *jitter* medido, artificialmente introduzido pelo facto de a aplicação receptora ficar por momentos impedida de receber. Para avaliar a influência deste efeito nas medições efectuadas, foram realizados testes com o mesmo conjunto de fluxos anteriores, em que a aplicação emissora é sempre corrida com escalonamento prioritário e a receptora é corrida com escalonamento normal e prioritário. Para melhor ilustrar o problema sem que o desempenho da rede afectasse os resultados (neste caso apenas se pretende avaliar a aplicação) os testes foram repetidos sobre a interface `lo` (*loopback*). As tabelas 5.3 e 5.4 mostram os resultados obtidos, com os tempos apresentados em μs.

		Fluxo 1			Fluxo 2		
		jitter médio	*jitter* máximo	pacotes perdidos	*jitter* médio	*jitter* máximo	pacotes perdidos
um só fluxo	normal	93	12198	0.06%	—	—	—
	prioritário	83	922	0	—	—	—
fluxos iguais	normal	94	15062	0.07%	95	14106	0.09%
	prioritário	85	1321	0	85	721	0
fluxos diferentes	normal	231	12863	0.1%	184	12837	0.11%
	prioritário	221	1629	0	175	1176	0

Tabela 5.3: Teste de recepção usando a rede

		Fluxo 1			Fluxo 2		
		jitter médio	*jitter* máximo	pacotes perdidos	*jitter* médio	*jitter* máximo	pacotes perdidos
um só fluxo	normal	15	14340	0	—	—	—
	prioritário	3	387	0	—	—	—
fluxos iguais	normal	16	14246	0	15	13349	0
	prioritário	3	229	0	2	129	0
fluxos diferentes	normal	15	12709	0	14	12962	0
	prioritário	3	82	0	3	275	0

Tabela 5.4: Teste de recepção na interface `lo`

Através da análise destas duas tabelas pode concluir-se que mesmo sem escalonamento prioritário no segundo caso não há perdas. Provavelmente isto deve-se ao facto de todo o processamento (nomeadamente o efectuado pelos controladores de dispositivo de rede) abaixo do nível IP ser eliminado neste caso. Por outro lado pode ainda concluir-se que a componente de *jitter* medido induzida pela aplicação é muito superior no caso de não ser utilizado escalonamento prioritário: superior a 10 ms em todos os testes com escalonamento normal, e da ordem de poucas centenas de microssegundos nos testes com escalonamento prioritário na interface `lo`[4].

[4]Não esquecer que no teste usando a rede o *jitter* introduzido pela própria rede é adicionado ao introduzido pela aplicação, tendo neste caso maior interesse os testes na interface `lo`.

5.2 Testes sem carga

Por forma a obter resultados conclusivos dos testes em carga é necessário dispor, como termo de comparação, dos resultados obtidos quando os fluxos de teste são injectados na rede sem qualquer outra carga. Nestas condições o atraso introduzido pela rede é mínimo, pois sendo a carga oferecida inferior ao débito da rede, os pacotes não vão ser atrasados em filas de espera. Assim, os resultados aqui obtidos representam um limite superior à qualidade de serviço expectável quando na rede em carga são introduzidos mecanismos de classificação e controlo de tráfego que visam obtê-la.

Na tabela 5.5 apresentam-se os resultados de três testes efectuados com um fluxo apenas, e outros três efectuados com dois fluxos[5], cada teste com a duração de 60 segundos. Os tempos na tabela são expressos em μs.

	Fluxo 1				Fluxo 2			
	atraso médio	*jitter* médio	*jitter* máximo	pacotes perdidos	atraso médio	*jitter* médio	*jitter* máximo	pacotes perdidos
1 fluxo	717	82	496	0	—	—	—	—
	717	83	512	0	—	—	—	—
	717	83	1989	0	—	—	—	—
2 fluxos	715	85	964	0	715	85	718	0
	715	85	1356	0	716	84	520	0
	715	85	1105	0	716	85	1106	0

Tabela 5.5: Teste de rede sem carga

Como pode observar-se, o atraso médio é constante ao longo dos testes e menor que 1 ms. O valor médio do *jitter* é também essencialmente constante, e menor que 100 μs. O valor de pico do *jitter*, por sua vez, apresenta uma variação apreciável, sendo o menor valor da ordem de 0.5 ms e o maior da ordem de 2 ms. Eventualmente, em testes de maior duração poderia ainda atingir valores algo superiores (embora a obtenção de um valor de pico de 1690 μs num teste com 20 min. de duração indique o contrário).

5.3 Latência abaixo do nível 3

Os testes efectuados até aqui basearam-se todos apenas nos fluxos de teste, de débito controlado e relativamente baixo, não existindo qualquer carga adicional na rede. Ao testar configurações e estados da rede mais próximos das condições normais de funcionamento, em que na maior parte das vezes existe oferta de tráfego suficiente para ocupar toda a capacidade de ligação, a existência de filas de espera abaixo do nível 3, nomeadamente no *device driver* e no próprio *hardware*, manifesta-se através da introdução de uma parcela adicional no atraso sofrido pelos pacotes, incluindo os de maior prioridade. Além do atraso, o facto de normalmente as filas de espera a este nível terem a

[5] A razão pela qual são apresentados três testes em vez de apenas um é a variabilidade dos resultados obtidos no que se refere ao *jitter* máximo.

capacidade limitada em número de pacotes e não em quantidade de dados dá também origem a um grande aumento do *jitter*.

5.3.1 Teste sem filas de espera

Para melhor compreender o fenómeno que acabou de ser descrito, foi efectuado um teste em que o escalonador é um FIFO, com uma fila de 0 pacotes. Na prática, isto significa que não existe fila: ou o pacote é transmitido mal chega ao sistema, ou então é eliminado. Com esta configuração foi gerado um fluxo com pacotes de 750 octetos e débito suficiente para esgotar a capacidade da ligação. Os resultados obtidos são os apresentados na tabela 5.6.

Débito	Atraso	*Jitter*	*Jitter* máximo
1720 kbps	114 ms	878 μs	4.56 ms

Tabela 5.6: Teste sem fila de espera

Numa ligação com uma capacidade de 2 Mbit/s[6], a transmissão de 750 octetos demora $\frac{750\times8}{2\times10^6} = 3 \times 10^{-3}$ segundos. Deixando uma margem de 10% para *overheads*[7], seria de esperar que o pacote sofresse um atraso de cerca de 3.3 ms. Claramente, os 114 ms de atraso obtidos diferem largamente desses 3.3 ms que seria de esperar. Dividindo o atraso médio pelo tempo de transmissão do pacote, chega-se à conclusão que existem cerca de 34 pacotes em espera. Os únicos sítios onde esta espera pode ocorrer são o *device driver* da placa ATM e o próprio *hardware*. De facto, a placa ATM utilizada[8] utiliza listas circulares para transmissão dos pacotes, que em fluxos CBR são de 64 elementos. Por cada AAL5-PDU transmitido, o *device driver* coloca duas entradas na lista, uma para identificar o pacote e outra para pedir uma interrupção no fim da sua transmissão. Assim, apesar de no nível 3 não existirem filas de espera, existindo oferta de tráfego suficiente estão permanentemente 32 pacotes em espera na fila no nível 2. Mesmo admitindo que a unidade máxima de transferência (MTU) é limitada a 1500 octetos[9], isto implica que o tráfego em serviço garantido pode ter que esperar mais de 200 ms apenas num *router*. Para a maior parte dos serviços com requisitos de tempo real, este valor é extremamente elevado[10], o que obriga a abordar o problema dos serviços garantidos de outra forma.

5.3.2 Testes com limitação de tráfego

Para confirmar a origem do atraso medido na secção anterior e no sentido de encontrar uma solução para o problema foi efectuado um teste em que uma disciplina de serviço TBF limita o débito a

[6] Capacidade bruta da ligação. Inclui os *overheads* até à camada AAL5, excluindo o da camada ATM.

[7] Cabeçalhos IP (20 octetos) e UDP (8 octetos), encapsulamento (8 octetos), e enchimento (*padding*) e informação (8 octetos) do AAL5. Em pacotes com 750 octetos de dados representam um *overhead* de 8.8%

[8] ForeRunner 155LE

[9] Valor comum em interfaces *ethernet*; em interfaces CLIP o valor padrão é de 9180 octetos.

[10] Por exemplo no caso de voz sobre IP, se existir um atraso total entre a emissão da voz e a sua reprodução superior a 250 ms o serviço é considerado de má qualidade. Neste caso, um atraso de 200 ms num só *router* é absolutamente inaceitável.

1 Mbit/s, com um tamanho de rajada de 1500 octetos. Sendo o débito após a filtragem inferior à capacidade da ligação, os pacotes que passam pela interface de rede encontram sempre a fila vazia ou praticamente vazia. O fluxo transmitido tem as mesmas características do utilizado no teste anterior.

Débito	Atraso	*Jitter*	*Jitter* máximo
990 kbps	8.51 ms	532 μs	2.81 ms

Tabela 5.7: Teste com limitação de tráfego

Como pode observar-se na tabela 5.7, o débito aproxima-se bastante do 1 Mbit/s configurado[11]. Mas o mais importante é que efectivamente houve uma grande redução no atraso médio sofrido pelos pacotes. Apesar de ainda ser mais do dobro do tempo que em teoria o pacote demoraria a ser transmitido, é necessario ter em conta que este tempo é contabilizado desde a emissão até à recepção, e não apenas o atraso sofrido no *router*.

5.3.3 Solução do problema e teste da solução

A abordagem usada para resolver o problema de latência descrito foi a alteração ao *device driver* da placa ATM para introduzir um limite ao número de entradas das referidas listas circulares para circuitos virtuais de categoria de serviço CBR. O limite introduzido foi de 4 entradas, o que significa que podem ficar no máximo 2 pacotes em espera.

Feita esta alteração, foi repetido o teste sem fila de espera, tendo desta vez sido obtidos os resultados que se apresentam na tabela 5.8.

Débito	Atraso	*Jitter*	*Jitter* máximo
1717 kbps	9.61 ms	751 μs	2.68 ms

Tabela 5.8: Novo teste sem fila de espera

Comparando os resultados das tabelas 5.6 e 5.8, torna-se evidente que a alteração introduzida solucionou o problema: o atraso foi reduzido por um factor superior a 10. Está, pois, demonstrada a influência que a placa de rede e o seu *device driver* podem ter na qualidade de serviço. Fica também demonstrado que a qualidade de serviço está dependente de todas as entidades envolvidas na transporte do pacote até ao seu destino.

5.4 Testes à disciplina de serviço CBQ

Sendo o CBQ a disciplina de serviço mais popularizada para partilha de ligações, é interessante observar como se comporta esta implementação. Nesta secção serão apresentados os resultados de diversos testes efectuados com vista a uma melhor compreensão essencialmente de pormenores menos óbvios de implementação, configuração e utilização.

[11] Dentro dos limites impostos pela granularidade temporal dos temporizadores, a disciplina TBF controla o débito de forma bastante precisa, ao contrário do que acontece com a disciplina CBQ, como se verá adiante.

5.4.1 Opções `bounded` e `isolated`

Segundo a definição dada em [15], a opção `bounded` (limitada) deve impedir a classe de utilizar capacidade excedente que a classe sua antecessora lhe pudesse ceder. Na prática, isto implica que uma classe `bounded` fica limitada à capacidade que lhe foi atribuída, ainda que exista um excedente que pudesse por ela ser utilizado. Quando aplicada a nós intermédios, no entanto, o resultado nem sempre é o esperado. Contudo, os resultados obtidos estão de acordo com as regras do CBQ, como se verá.

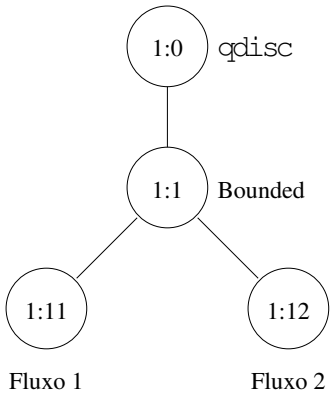

Figura 5.1: Configuração CBQ para o teste à opção `bounded`

A configuração usada, ilustrada na figura 5.1, consiste numa classe (1:1), que é sempre `bounded`, e duas classes suas descendentes (1:11 e 1:12). Nos diversos testes foram usados diferentes valores de capacidade para cada uma das classes, bem como a utilização da opção `bounded` nas duas classes-folha. Todas as classes têm a mesma prioridade. O *script* básico para configuração é apresentado no apêndice E.1. Os fluxos de teste consistem em pacotes UDP com 64 octetos de dados.

A tabela 5.9 mostra os resultados obtidos nos diversos testes. Tanto os resultados como a capacidade atribuída a cada ligação são apresentados em kbps. A letra "B" indica que a classe é `bounded`.

Os resultados dos testes 1 e 5 mostram que não é possível apenas com CBQ criar uma regra em que uma dada classe não possa exceder um débito x e cada uma das suas descendentes não possa exceder, respectivamente, y e z, sendo $x < y + z$. Neste caso, as subclasses irão ter débito total de $y + z$, excedendo o valor x. Embora este comportamento possa não ser óbvio, ele está inteiramente de acordo com as regras do CBQ, segundo as quais uma classe (folha) só pode ser regulada se estiver a transmitir acima do seu limite. O teste 2 confirma-o: enquanto a capacidade afecta à classe que transporta o fluxo é inferior à da sua antecessora, o limite prevalecente é o da antecessora, pois apenas esta é `bounded`; quando a capacidade da classe que transporta o fluxo é superior à da sua antecessora `bounded`, só quando o tráfego excede o valor da sua própria capacidade é que o seu débito é regulado.

	Classe			Resultados		
Teste	1:1	1:11	1:12	fluxo 1	fluxo 2	total
1	100 B	50 B	—	53	—	53
	100 B	100 B	—	110	—	110
	100 B	150 B	—	166	—	166
2	100 B	50	—	106	—	106
	100 B	100	—	110	—	110
	100 B	150	—	161	—	161
3	100 B	50	150	56	165	221
	200 B	50	150	56	169	225
	300 B	50	150	85	253	338
4	300 B	50 B	150	57	279	336
	300 B	50	150 B	167	168	335
5	100 B	50 B	150 B	57	168	225
	300 B	50 B	150 B	57	168	225
6	100 B	50 B	25	54	53	107
	100 B	50 B	50	55	55	110
	100 B	50 B	75	55	82	137

Tabela 5.9: Testes à opção `bounded`.

Os testes 3 e 4 mostram que se a capacidade de uma classe `bounded` for superior à das suas descendentes com tráfego, as descendentes não-`bounded` vêem o seu débito aumentar proporcionalmente à capacidade que lhes está atribuída, até atingir o limite da antecessora `bounded`. Também isto está de acordo com os objectivos de partilha de ligação da disciplina CBQ.

Em suma, se a folha for `bounded`, o seu débito é independente do limite das suas antecessoras. Se a folha não for `bounded` mas a sua antecessora for, pode sempre transmitir pelo menos até ao seu limite, podendo ainda usar um excedente de capacidade proporcional à capacidade que lhe está atribuída, até atingir o limite da antecessora.

É de salientar que estes resultados apenas se aplicam ao caso em que não existe nenhum tráfego adicional no sistema. Assim, não existe nenhuma classe insatisfeita, tornando desnecessário regular classes não-`bounded`.

Em [15], uma classe `isolated` (isolada) define-se como uma classe que não excede a sua capacidade nem deixa que outras usem a capacidade que lhe está atribuída e não é usada: simplesmente é-lhe atribuída uma fracção da capacidade da ligação. Esta definição é falaz, pois assume a existência de um meio externo ao CBQ de atribuir uma fracção da capacidade da ligação. Os testes efectuados demonstram que pelo menos nesta implementação do CBQ a opção `isolated` não tem o efeito esperado. De facto, os resultados apenas mostram duas coisas:

1. O tráfego de uma classe `isolated` não é contabilizado para as suas antecessoras, o que pode também comprovar-se correndo no *router* o comando `tc -s class sh dev atm3`. Assim, mesmo que tenha uma antecessora `bounded` uma classe `isolated` não é regulada enquanto

houver capacidade disponível[12].

2. O facto de usar a opção `isolated` numa folha da hierarquia tem implicações nefastas no funcionamento de suas antecessoras que sejam `bounded`, nomeadamente levando à obtenção de resultados muito variávies mesmo quando o tráfego oferecido é determinístico com débito constante, variabilidade essa que se estende a outras classes descendentes da mesma antecessora bounded.

Este segundo ponto deve-se, provavelmente, a algum *bug* (erro) ou limitação de ordem prática na implementação da disciplina CBQ.

Para testar o efeito da opção `isolated`, bem como a sua interacção com classes `bounded`, foi criada a seguinte estrutura: a raiz da hierarquia (1:0), com uma capacidade igual ao total da ligação (2 Mbps); desta classe descendem duas outras, 1:1 e 1:2, em que 1:1 tem uma capacidade de 500 kbps e é `bounded` e 1:2 tem 1.5 Mbps; a classe 1:1 tem duas descendentes, 1:11 e 1:12, ambas com 250 kbps, sendo 1:12 `isolated`; a classe 1:2 tem também duas descendentes, 1:21 e 1:22, cada uma com 750 kbps. Todas estas classes têm a mesma prioridade. Os fluxos de tráfego gerados consistem em datagramas UDP com 750 octetos de dados, sendo os fluxos 1 a 4 encaminhados, respectivamente, para as classes 1:11, 1:12, 1:21 e 1:22. O *script* básico usado para gerar esta configuração é apresentado no apêndice E.2.

Teste	Fluxo	Débito (kbps)	Atraso médio	*Jitter* médio	*Jitter* de pico
1 fluxo sem CBQ	1	1717	357 ms	758 μs	2.69 ms
só fluxo 1 com CBQ	1	578	1.04 s	15.0 ms	16.6 ms
só fluxo 2 com CBQ	2	1717	357 ms	759 μs	2.73 ms
fluxos 1 e 2 sem CBQ	1	860	358 ms	525 μs	6.28 ms
	2	858	358 ms	524 μs	5.33 ms
fluxos 1 e 2 com CBQ	1	479	1.25 s	16.1 ms	33.5 ms
	2	480	1.25 s	18.9 ms	44.5 ms
Todos os fluxos com CBQ	1	345	1.73 s	13.9 ms	304 ms
	2	271	2.23 s	16.8 ms	72.9 ms
	3	555	1.08 s	4.31 ms	300 ms
	4	553	1.09 s	4.52 ms	316 ms
Todos os fluxos com CBQ (2)	1	278	2.15 s	16.5 ms	354 ms
	2	278	2.27 s	21.6 ms	204 ms
	3	585	1.03 s	4.67 ms	345 ms
	4	583	1.03 s	5.37 ms	366 ms

Tabela 5.10: Testes à opção `isolated`

A tabela 5.10 mostra os resultados da primeira série de testes efectuada. Da comparação de resultados entre o teste com um fluxo e sem CBQ e o teste com apenas o fluxo 2 e com CBQ pode concluir-se que independentemente de uma antecessora sua ser `bounded`, uma classe `isolated` não é

[12] E, portanto, não existirem classes insatisfeitas.

regulada se houver capacidade disponível. Em contraste com isto, o resultado do teste apenas com o fluxo 1 e com CBQ mostra que se a classe não for `isolated`, está sujeita a limitação imposta pelo facto de a sua antecessora ser `bounded`, muito embora a regulação seja pouco precisa, conduzindo a um débito de 578 kbps em vez dos 500 kbps configurados.

No caso em que existe tráfego suficiente para utilizar toda a capacidade da ligação e em todas as classes, pelas regras de partilha de ligação do CBQ seria de esperar que as várias classes tivessem um débito aproximado da sua capacidade configurada, pois o facto de a classe ser isolada não implica que esteja isenta das regras de partilha de ligação. No segundo teste com todos os fluxos é isto que acontece, embora as classes de menor débito tenham sido beneficiadas em detrimento das de maior débito. No entanto este desempenho nem sempre se verifica: no primeiro teste o fluxo 1 (correspondente à classe 1:11) excede largamente a sua capacidade configurada, equanto os fluxos 3 e 4 (correspondentes às classes 2:11 e 2:12) recebem bastante menos do que deviam. E se esta variabilidade de resultados existe neste ambiente de teste controlado em que os pacotes são emitidos a uma cadência constante, em princípio num ambiente real a variabilidade seria ainda maior. Poderia pensar-se que esta variabilidade se deve ao facto de a relativamente curta duração dos testes (60 s) afectar a credibilidade dos resultados, mas em testes de maior duração (5 min.) foram obtidos resultados semelhantes, pelo que esta hipótese não é credível.

O teste com os fluxos 1 e 2 com CBQ tem um resultado surpreendente: apesar de a soma dos débitos das duas classes com tráfego ser bastante inferior à capacidade total da ligação, a classe `isolated` está a ser regulada. Além disso, a classe 1:11 vê o seu débito regulado para um valor inferior ao obtido quando o fluxo 2 não existe. De facto, neste teste seria de esperar que o fluxo 1 ficasse limitado à capacidade da antecessora `bounded` da classe 1:11 (de facto, um pouco mais, pois havendo excedente de capacidade o CBQ tem tendência a regular o tráfego para valores acima dos configurados) e o fluxo 2 ocupasse toda a restante capacidade. No entanto, ambas são reguladas para um mesmo valor, um pouco inferior à capacidade da sua antecessora `bounded`.

Para tentar compreender melhor este fenómeno foram realizados dois grupos de testes adicionais: primeiro variando o tráfego oferecido na classe 1:12 de 0 até um valor superior à capacidade total da ligação, mantendo na classe 1:11 a oferta de tráfego constante e também com um valor superior à capacidade total da ligação; posteriormente usando um procedimento simétrico (classe 1:12 com oferta de tráfego constante e classe 1:11 com oferta de tráfego variável). Estes testes foram, no entanto, inconclusivos, dada a variabilidade obtida nos resultados de repetições do mesmo teste. Isto apesar de os fluxos emitidos serem de débito constante, portanto com caracteerísticas perfeitamente determinísticas. Fica, assim, apenas a ideia de que a opção `isolated` induz alguma instabilidade no funcionamento da disciplina de serviço CBQ.

Um conjunto adicional de testes, cujos resultados se apresentam na tabela 5.11, permite obter algumas conclusões adicionais. No primeiro destes testes foi retirada a opção `isolated` à classe 1:12. Como seria de esperar, neste caso o débito conjunto das duas classes é limitado ao da sua antecessora `bounded`[13], sendo equitativamente distribuído por elas. Ao contrário dos testes

[13]Débito total de 608 kbps, correspondente aos 500 kbps configurados mais uma margem por excesso, sempre

anteriores, os resultados obtidos em diversas repetições deste teste foram sempre aproximadamente os mesmos.

Teste	Fluxo	Débito (kbps)	Atraso médio	*Jitter* médio	*Jitter* de pico
fluxos 1 e 2, classe 1:12 não `isolated`	1	304	1.96 s	14.6 ms	67.5 ms
	2	304	1.96 s	14.6 ms	67.5 ms
fluxos 1 e 2, classe 1:12 `bounded isolated`	1	609	989 ms	13.4 ms	20.1 ms
	2	306	1.95 s	3.80 ms	22.5 ms
fl. 1 e 2, classe 1:12 controlada por TBF	1	606	993 ms	13.9 ms	19.4 ms
	2	248	318 ms	2.44 ms	27.7 ms

Tabela 5.11: Testes adicionais à opção isolated

No segundo teste desta série a classe 1:12 foi configurada como `bounded` além de `isolated`. Neste caso o débito da classe 1:12 é regulado para a capacidade que lhe está atribuída (embora com a margem de erro por excesso usual), e a classe 1:11 é regulada para a capacidade da sua antecessora `bounded` (com a margem de erro proporcional). Neste caso, os resultados são os previsíveis, e repetições do teste revelam a sua consistência.

Por fim, foi efectuado um teste em que a classe 1:12 volta a ser apenas `isolated`, mas o seu débito é controlado instalando uma disciplina de serviço TBF como descendente da classe 1:12. Os resultados deste teste são muito semelhantes aos do anterior, com a excepção de dois factores:

1. A classe 1:12 é regulada de forma muito mais precisa, tendo um débito efectivo praticamente igual à capacidade configurada. Conclui-se, pois, que a disciplina TBF, desde que se respeitem os limites mencionados na secção 4.2.7, é bastante superior ao CBQ para limitação de tráfego.

2. Existe uma marcada redução no atraso sofrido pelos pacotes do fluxo 2. Isto deve-se ao facto de a disciplina TBF instalada como descendente da classe 1:12 ter uma fila limitada a 10 koctetos, em vez da limitação a 100 pacotes[14] usada pela disciplina de serviço genérica, instalada por omissão em todas as classes.

Dos resultados obtidos nos conjuntos de testes anteriores é possível concluir que:

1. Se é estritamente necessário limitar o débito de uma classe folha mesmo quando não há tráfego, é preferível usar uma disciplina de serviço TBF como descendente da classe a limitar do que apenas declarar essa classe `bounded`. Se a limitação imposta for apenas para impedir uma classe de monopolizar a ligação, então declarar a classe `bounded` já é suficiente.

2. É preferível não usar a opção `isolated`, particularmente em classes que tenham ascendentes `bounded`.

Sobre o CBQ, um outro facto que deve ser mencionado é o seguinte: durante a realização destes testes verificou-se que em alguns casos havia um pacote que ficava pendente no router no final de

introduzida pelo CBQ.

[14] 100 pacotes de 750 octetos = 75000 octetos.

um teste e que só era efectivamente transmitido quando começava a ser gerado tráfego do teste seguinte. A detecção deste efeito deve-se ao facto de em alguns testes aparecer um *jitter* de pico superior a 100 segundos, e todos os pacotes excepto um estarem fora de sequência: o último pacote de um teste só era efectivamente transmitido quando se iniciava o seguinte[15]. Assim, a diferença entre o tempo de transmissão desse pacote e o primeiro do teste seguinte era enorme, e como o seu número de sequência era superior ao dos pacotes recebidos, estes eram vistos como chegando fora de ordem. Este comportamento indicia um problema na gestão de temporizadores, pois o temporizador que devia despoletar a transmissão deste pacote não está a ser accionado[16].

5.4.2 Serviços garantido, prioritário e tipo melhor esforço

Para testar a possibilidade de coexistência de diferentes serviços numa ligação usando CBQ, foi configurado um cenário com três serviços:

1. Serviço garantido, de elevada prioridade, para tráfego com requisitos apertados de tempo-real e que se supõe sujeito a controlo de admissão.

2. Serviço prioritário, com melhor desempenho que um serviço tipo melhor esforço mas sem qualquer garantia.

3. Serviço tipo melhor esforço, com prioridade inferior aos dois anteriores.

Neste sentido, foram criadas três classes, 1:1, 1:2 e 1:3. A classe 1:1 destina-se a transportar fluxos de serviço garantido, tendo uma capacidade configurada de 250 kbit/s, com limitação. A sua prioridade é 1. A classe 1:2, destinada ao transporte do tráfego prioritário, tem uma capacidade de 750 kbit/s, também limitada, e com prioridade 2. A classe para tráfego de tipo melhor esforço, 1:3, ocupa os restantes 1000 kbit/s da ligação. Esta classe é de prioridade 3, a mais baixa. Assim, não é necessário impor uma limitação, pois não vai impedir nenhuma das outras classes de transmitir. Os fluxos 1, 2 e 3 são encaminhados, respectivamente, para as classes 1:1, 1:2 e 1:3. A tabela 5.12 mostra as características dos fluxos gerados e os resultados obtidos, sendo o débito indicado em kbit/s.

Tráfego oferecido				Resultados				
Fl.	Tam. Pac.	Pac./s	Débito	Débito	Atraso	*Jitter*	*Jit.* max.	Perdas
1	64	400	205	205	4.30 ms	2.22 ms	6.70 ms	0
2	750	120	720	720	8.58 ms	2.35 ms	7.61 ms	0
3	750	500	3000	473	1.28 s	4.44 ms	20.6 ms	84%

Tabela 5.12: Diferenciação de serviços com CBQ

[15]Estes resultados não se mostram aqui, pois sempre que isso acontecia o teste era repetido por forma a obter resultados relevantes para o que está em causa: as opções `bounded` e `isolated`.

[16]Logicamente, este efeito não se manifesta enquanto houver outro tráfego, que ao gerar interrupções invoca as funções de gestão de filas do CBQ sem necessidade de temporizadores.

Os parâmetros de maior interesse neste caso são o débito e as perdas, pois tratando-se de fluxos sintéticos com tamanhos de pacote e débitos constantes os valores medidos de *jitter* têm pouco significado no caso do serviço prioritário e nenhum para a classe de serviço tipo melhor esforço. Pode observar-se que sendo o tráfego oferecido às classes de serviço garantido e prioritário inferiores às respectivas capacidades configuradas, o atraso sofrido pelos pacotes destas classes é bastante baixo, sendo apenas elevado na classe de serviço tipo melhor esforço, pois nesta classe existe um permanente congestionamento. Neste caso, uma vez que o escalonador de partilha não chega a regular nenhuma das classes e existe apenas uma classe em cada prioridade, o comportamento é idêntico ao de um simples escalonador de prioridade estrita. De facto, efectuando o mesmo teste usando a disciplina `prio`, os resultados obtidos são idênticos.

Em contraste com estes resultados, a tabela 5.13 mostra os resultados de um teste idêntico efectuado antes da solução do problema de latência a nível do *device driver*. Note-se como neste caso não só são afectados os atrasos e o *jitter*, como também são introduzidas perdas na classe de tráfego prioritário, apesar de o tráfego oferecido ser inferior à sua capacidade configurada. Isto parece indicar que a existência de filas relativamente longas no *device driver* e/ou *hardware* de algum modo "confunde" a disciplina de serviço CBQ, que de alguma forma regula a classe 1:2, levando a sua fila a aumentar até ao limite.

Tráfego oferecido				Resultados				
Fl.	Tam. Pac.	Pac./s	Débito	Débito	Atraso	*Jitter*	*Jit.* max.	Perdas
1	64	400	205	205	55.3 ms	2.61 ms	12.1 ms	0
2	750	120	720	565	1.06 s	6.99 ms	38.2 ms	20.6%
3	750	500	3000	634	1.00 s	13.9 ms	92.5 ms	78.6%

Tabela 5.13: Diferenciação de serviços com CBQ antes da resolução do problema de latência no *device driver* ATM

Um aspecto que chama a atenção é o facto de, tanto no primeiro caso como no segundo, o débito total ser bastante inferior à capacidade configurada de 2 Mbit/s. De facto, o débito total na tabela 5.12 é de apenas 1398 kbit/s. Este valor refere-se ao débito útil, mas calculando o *overhead* das diversas camadas é possível calcular o débito bruto, possibilitando a comparação com a capacidade configurada.

Além dos dados, cada pacote contém:

- 8 octetos de cabeçalho UDP
- 20 octetos de cabeçalho IP
- 8 octetos de encapsulamento LLC/SNAP
- 8 octetos de informação AAL5
- Enchimento AAL5 por forma a que o tamanho total seja múltiplo de 48 octetos, preenchendo um número inteiro de células ATM

Assim, um pacote com 64 octetos de dados fica com um total de 144 octetos, e um pacote com 750 octetos de dados fica com um total de 816 octetos, correspondendo a *overheads* de, respectivamente, 125% e 8.8%. Assim, o valor de débito bruto é $205 \times 2.25 + (565 + 634) \times 1.088 \simeq 1766\,\text{kbit/s}$. Ainda assim este valor é bastante inferior aos 2 Mbit/s configurados, e uma vez que esta capacidade é por si bastante baixa, seria impossível atribuir este abaixamento de débito apenas a limitações da CPU.

Uma hipótese seria a de estar a ser inserida uma célula ATM adicional por cada pacote. Assim, os overheads passariam a ser de 200% e 15.2%, e o débito bruto subiria para $205 \times 3 + (565 + 634) \times 1.152 \simeq 1996\,\text{kbit/s}$, um valor muito aproximado dos 2 Mbit/s configurados. De igual modo, o débito bruto obtido apenas com pacotes de 750 octetos no teste sem fila de espera (ver tabela 5.6) seria de $1720 \times 1.152 \simeq 1981\,\text{kbit/s}$, em vez dos 1871 kbit/s obtidos considerando o *overhead* normal de apenas 8.8%.

Esta hipótese está aparentemente correcta, devendo-se o desperdício de uma célula por cada pacote enviado a uma limitação do controlador SAR da placa ATM utilizada que não é devidamente compensada pelo *device driver*. De cada vez que é pedida uma interrupção de fim de transmissão de pacote ao controlador SAR ele perde a oportunidade de enviar uma célula para processar esse pedido. Em circuitos virtuais com categoria de serviço CBR é efectuado um pedido de interrupção por cada pacote enviado, o que leva a uma perda de débito correspondente a uma célula por pacote.

Para confirmar esta hipótese foi efectuado um novo teste, alterando o *device driver* ATM por forma a que apenas fosse gerado um pedido de interrupção por cada 9 pacotes transmitidos[17]. Os resultados apresentam-se na tabela 5.14.

Tráfego oferecido				Resultados				
Fl.	Tam. Pac.	Pac./s	Débito	Débito	Atraso	*Jitter*	*Jit.* max.	Perdas
1	64	400	205	205	13.0 ms	1.77 ms	14.6 ms	0
2	750	120	720	720	14.3 ms	3.68 ms	14.2 ms	0
3	750	500	3000	655	931 ms	5.93 ms	15.0 ms	79%

Tabela 5.14: Teste para confirmação do problema de quebra de débito

Usando os valores normais de *overhead* para calcular o débito bruto, obtém-se um valor de $205 \times 2.25 + (720 + 655) \times 1.088 \simeq 1957\,\text{kbit/s}$, muito mais próximo do valor configurado de 2 Mbit/s. Torna-se assim clara a existência de um problema que com tamanhos de pacote suficientemente grandes poderia passar despercebido.

5.4.3 Filas em CBQ

Em qualquer um dos testes efectuados ao CBQ onde existam classes é notória a introdução de enormes valores de atraso no tráfego pertencente a classes onde existem perdas, isto é, aquelas onde a fila enche. De facto, na tabela 5.12, por exemplo, o tráfego de tipo melhor esforço sofre um

[17] Para poder efectuar esta alteração foi necessário permitir que a fila de espera no nível 2 aumentasse para 10 pacotes, pelo que apenas o presente teste foi efectuado com estas alterações.

atraso de 1.28 segundos. Outros exemplos são os testes com fluxos 1 e 2 e com todos os fluxos na tabela 5.10, onde os atrasos chegam a exceder 2 segundos. Estes valores de atraso, inversamente proporcionais ao débito efectivo da classe, devem-se ao facto de por omissão as classes CBQ usarem disciplinas de serviço genéricas para armazenar os pacotes até ao instante da sua transmissão. A disciplina de serviço genérica instala três filas limitadas em número de pacotes (embora normalmente apenas uma seja utilizada). Neste tipo de interfaces (*ethernet* e CLIP) as filas têm um limite de 100 pacotes. Este limite, embora seja perfeitamente adequado a interfaces 100baseTX que, com um valor da MTU de 1500 octetos, se traduz num atraso de cerca de 12 ms, já não o é no caso de interfaces *ethernet* a 10 Mbps em meio partilhado, em que o limite inferior do atraso máximo é de cerca de 120 ms, e é totalmente inaceitável para classes com capacidade da ordem de 1 Mbps ou menos. O facto de o valor padrão da MTU em interfaces CLIP ser de 9180 octetos, ou seja, cerca de 6 vezes superior ao caso da *ethernet*, agrava pelo mesmo factor de 6 vezes o atraso sofrido no pior caso.

Num percurso que atravesse 15 *routers*, um atraso de 1 s em cada *router* traduz-se num atraso total de 15 s, ou seja, 30 segundos de tempo de ida e volta (RTT). Obviamente, atrasos desta magnitude impossibilitam o uso de qualquer aplicação de rede interactiva.

Para solucionar este problema, é necessário instalar como descendentes das classes CBQ (bem como em quaisquer outras que usem a disciplina de serviço genérica para armazenar os pacotes) disciplinas de serviço que permitam configurar o tamanho da fila, cujos casos mais simples são `pfifo` ou `bfifo`. Em classes que possam sofrer algumas perdas, disciplinas de serviço como `red` ou `gred` têm a vantagem adicional de as introduzir de forma gradual, conduzindo a um "ponto de funcionamento" mais ou menos estável pelo menos para tráfego TCP. Em alternativa, se o número de fluxos simultâneos for relativamente pequeno (até mais ou menos 100 fluxos), para o tráfego tipo melhor esforço pode ser interessante usar como folha uma disciplina de serviço `sfq`, pois em relação à família RED apresenta a vantagem de controlar também os fluxos UDP. Se for possível garantir à classe um valor de débito (em redes com meio partilhado isso não é possível), uma alternativa é a instalação de policiamento com um débito um pouco inferior ao garantido e um `burst` convenientemente dimensionado. Esta última alternativa é principalmente válida em classes de tempo-real em que a existência de perdas é menos grave do que valores elevados de atraso.

5.5 *Routers diffserv*

Nesta secção serão apresentados *scripts* para a configuração de *routers* com suporte para serviços diferenciados, implementando os PHB AF e EF, bem como uma classe para tráfego tipo melhor esforço.

5.5.1 Nó interior (*core*)

O apêndice F.1 contém o *script* usado para configurar um nó interior de um domínio *diffserv*, contendo diversas variáveis que correspondem a parâmetros configuráveis, nomeadamente as capa-

cidades atribuídas aos vários serviços, os tamanhos das filas, os limiares de probabilidades de perda RED.

Na raiz da hierarquia encontra-se uma pseudo-disciplina de serviço `dsmark` (`1:0`), responsável pela inicialização do campo `skb->tc_index` com o octeto DS do cabeçalho IP, bem como por guardar neste campo o resultado do filtro `tcindex` nela instalado (`parent 1:0`). Este filtro usa uma máscara 0xfc e um desvio (*shift*) de 2 *bits* para seleccionar apenas o campo DSCP dentro do octeto. Os vários elementos deste filtro separam o tráfego EF e o tráfego das quatro classes AF e as suas probablilidades de perda. As partes inferiores dos parâmetros `classid` destes elementos são estruturadas por forma a conter nos 4 *bits* menos significativos (ordem 0 a 3) a probabilidade de perda AF e nos 4 *bits* de ordem 4 a 7 a parte inferior do identificador da classe `2:x` à qual serão posteriormente atribuídos. Uma vez que é usada a opção `pass_on` no filtro `tcindex`, este não tenta efectuar o mapeamento algorítmico, pelo que todo o tráfego não seleccionado é marcado com o valor 1 (`default_index` de `dsmark`).

Como descendente de `1:0`, é configurada uma disciplina de serviço `cbq` (`2:0`), responsável pela efectiva gestão de recursos entre as classes de tráfego. As classes `2:1` a `2:4`, de prioridade 2, destinam-se a transportar o tráfego das classes AF1 a AF4. A classe `2:5`, de prioridade 1 e isolada, destina-se ao transporte do tráfego EF. Por fim, a classe `2:6` é usada para transportar o tráfego de tipo melhor esforço. Estas classes são seleccionadas pelo filtro `tcindex` (`parent 2:0`). A máscara 0xf0 e o desvio de 4 *bits* permitem isolar os 4 *bits* de `skb->tc_index` que contêm a selecção de classe, o que permite aos elementos específicos efectuar um mapeamento directo.

Como descendente da classe `2:5` (EF) é instalada uma disciplina de serviço `tbf` (`25:0`), para garantir que a classe nunca excede o débito que lhe foi atribuído, evitando assim ser regulada pelo escalonador de partilha CBQ, com os efeitos adversos que isso teria no atraso e no *jitter* que implicariam a perda de conformidade com o PHB EF [11]. As classes `2:1` a `2:4` (AF) têm como descendentes instâncias da disciplina de serviço `gred`, cuja prioridade de perda é seleccionada pelos 4 *bits* menos significativos de `skb->tc_index`. Para que efectivamente as probabilidades de perda das classes AFxy sejam crescentes com y em todas as situações, em conformidade com [10], é usado o esquema `grio`.

Para o tráfego tipo melhor esforço é utilizada como folha uma disciplina de serviço `red`, que aqui desempenha o duplo papel de reduzir o tamanho da fila em relação ao que teria uma disciplina de serviço genérica e de estabilizar o "ponto de funcionamento" dos fluxos TCP.

Para testar esta configuração foi efectuado um teste em que é oferecido à classe EF tráfego um pouco abaixo da capacidade configurada, às classes AFxy tráfego um pouco acima da sua capacidade, e à classe BE (melhor esforço) tráfego excedendo bastante a sua capacidade. Todos os fluxos consistem em datagramas UDP de 750 octetos. Os resultados apresentam-se na tabela 5.15.

Em termos de débito não se verificam quaisquer surpresas: na classe EF não existiram perdas, uma vez que o tráfego oferecido estava um pouco abaixo da capacidade configurada; nas classes AFxy as perdas aumentaram com o aumento de y, em conformidade com [10]; os somatórios em y

Fluxo	Prob. AF	Capac. (kbit/s)	Tr. ofer. (kbit/s)	Débito (kbit/s)	Atraso (ms)	*Jitter* (ms)	*Jit.* max. (ms)	Perdas
EF	—	250	228	228	8.89	1.74	2.38	0
AF1	1	125	48	47.9	418	31.7	158	0.2%
	2		48	46.1	462	31.0	198	3.75%
	3		48	30.4	490	35.5	251	36.4%
AF2	1	125	48	47.9	408	28.7	166	0.2%
	2		48	46.3	452	29.5	152	3.54%
	3		48	30.9	478	34.1	342	35.6%
AF3	1	125	48	48	413	31.7	137	0
	2		48	46.1	456	29.6	152	3.99%
	3		48	30.9	482	32.9	252	35.6%
AF4	1	125	48	48	409	27.0	122	0
	2		48	46.1	454	29.2	158	3.75%
	3		48	30.6	479	33.6	294	36.0%
BE	—	1250	1800	992	128	4.32	94.5	44.7%

Tabela 5.15: Nó interior: teste 1

do débito das classes AFxy é practicamente igual capacidade configurada para todos os grupos AFx; apenas o débito do tráfego tipo melhor esforço fica aquém da capacidade configurada, mas também isto era previsível, pois a classe BE é de prioridade inferior às classes EF e AF, e já anteriormente se verificou que a capacidade útil da ligação com pacotes deste tamanho se quedava nos 1717 kbit/s (aqui o somatório dos débitos ascende a 1720 kbit/s).

Um resultado aparentemente surpreendente é o de o atraso sofrido pelos pacotes das classes AF ser bastante superior ao sofrido pelos pacotes da classe BE. No entanto é preciso não esquecer que cada grupo AF tem uma fila de espera própria, independente da fila da classe BE. Por outro lado, o tamanho das filas não é muito diferente, ao passo que a capacidade atribuída[18] à classe BE é 10 vezes superior. Tendo em conta estes factores, e uma vez que neste teste existe congestionamento nas classes AF[19], seria de esperar que o atraso sofrido fosse superior ao da classe BE.

Foi efectuado um segundo teste em que o débito nas classes AF foi reduzido para um valor um pouco inferior à sua capacidade configurada, e cujos resultados se apresentam na tabela 5.16. Aqui, devido à ausência de congestionamento, as classes AF não registaram perdas, e o atraso médio diminuiu substancialmente. No entanto, a diferença entre o atraso sofrido por pacotes com diferentes probabilidades de perda dentro da mesma classe AF poderia induzir a conclusão errónea de que existe reordenação de pacotes dentro de um mesmo grupo AF. Tal não se verifica, e a explicação para este comportamento é simples. De facto, trata-se de um teste sintético, com tráfego gerado independentemente para cada fluxo. Acontece que, tendo o mesmo débito, os pacotes das 3 probabilidades de perda dentro de cada grupo AF eram gerados de forma quase simultânea. Assim, a disciplina de serviço CBQ podia enviar dois pacotes (correspondentes a AFx1 e AFx2) de uma vez (`allot`=1508), forçando o terceiro pacote a esperar. Dado que o tempo nominal para

[18] Embora não seja a que na prática é conseguida.

[19] Que, por se tratar de tráfego UDP, o mecanismo GRED não consegue controlar.

Fluxo	Prob. AF	Capac. (kbit/s)	Tr. ofer. (kbit/s)	Débito (kbit/s)	Atraso (ms)	*Jitter* (ms)	*Jit.* max. (ms)	Perdas
EF	—	250	228	228	8.89	1.74	2.36	0
AF1	1	125	36	36.1	9.12	0.84	4.55	0
	2		36	36.1	18.1	14.3	26.9	0
	3		36	36.1	76.7	17.2	57.0	0
AF2	1	125	36	36.1	14.1	2.14	9.41	0
	2		36	36.1	14.3	3.94	18.5	0
	3		36	36.1	73.6	18.5	46.5	0
AF3	1	125	36	36.1	20.8	3.74	12.9	0
	2		36	36.1	25.3	8.40	19.9	0
	3		36	36.1	92.7	17.3	60.4	0
AF4	1	125	36	36.1	26.8	4.38	12.9	0
	2		36	36.1	27.4	4.10	12.9	0
	3		36	36.1	60.2	24.8	67.5	0
BE	—	1250	1800	1058	121	4.08	43.2	41.2%

Tabela 5.16: Nó interior: teste 2

transmissão de um pacote de 750 octetos com um débito de 125 kbit/s é de cerca de 50 ms, pode concuir-se que a diferença de atrasos não implica qualquer reordenação. A repetição deste teste com a alteração do instante de activação dos fluxos por forma a os pacotes das 3 probabilidades de perda de cada grupo AF ficarem espaçados por intervalos de tempo aproximadamente iguais revela isto mesmo: os atrasos sofridos têm valores muito mais próximos. Isto revela a importância que tem o planeamento cuidado dos testes sintéticos a efectuar para que factores desta natureza não influenciem os resultados e, consequentemente, as conclusões que deles se podem retirar.

Poderia pensar-se que o problema que se acabou de descrever poderia igualmente ter originado o facto de no primeiro teste as classes AF com maior probabilidade de perda terem sofrido um maior volume de perdas. No entanto a repetição do teste com o referido espaçamento entre a geração dos pacotes deu origem a resultados em tudo idênticos aos da tabela 5.15, pelo que esta hipótese tem que ser rejeitada.

5.5.2 Nó fronteira

Na fronteira de um domínio *diffserv*, os *routers* devem dispor de toda a funcionalidade dos nós interiores. Adicionalmente, estes nós devem implementar funções de formatação de tráfego e/ou policiamento por forma a garantir a conformidade dos fluxos que entram no domínio com o TCA. Estes nós podem ainda efectuar a marcação de pacotes originados em máquinas que não suportem essa funcionalidade, permitindo que também os seus utilizadores possam beneficiar de serviços diferenciados. Esta marcação deve ser configurável administrativamente. Por este motivo, o *script* de configuração para um nó fronteira, apresentado no apêndice F.2, é mais complexo que o de um nó interior.

Na interface de ingresso está configurado um conjunto de filtros `u32` utilizados quer para atribuir

a um PHB tráfego originário de máquinas incapazes de efectuar elas próprias a marcação, quer para inicializar o campo `skb->tc_index` para pacotes previamente marcados. No filtro instalado com `pref` 1 existem elementos específicos que exemplificam a primeira função. Assim, o tráfego originado na porta 3025 (0x0bd1) da máquina cujo IP é 192.168.193.96 e que não exceda 50 kbit/s é atribuído ao PHB EF, sendo o excesso eliminado. O tráfego destinado à máquina 192.168.225.97 e que não exceda 30 kbit/s é atribuído ao PHB AF21; no entanto, o excesso, em vez de ser eliminado, é passado ao filtro seguinte, instalado com `pref` 2. Deste excedente, a parte que não exceda outros 30 kbit/s é repescada para o PHB AF22, sendo o seu excedente repescado pelo filtro com `pref` 3 para o PHB AF23, até 40 kbit/s. O tráfego que exceda este último limite é eliminado.

Com `pref` 10 estão instalados os elementos que permitem inicializar o campo `skb->tc_index` para pacotes com o DSCP previamente marcado. Após esta sequência de filtros, os dois *bits* menos significativos de `skb->tc_index` vão conter a probabilidade de perda para o grupo AF. Os três *bits* seguintes vão conter a classe AF, o valor 5 para EF ou 0 para o tráfego tipo melhor esforço. O *bit* de ordem 8 vai ficar a 1 nos pacotes AFx3, sendo usado para permitir o policiamento e reclassificação *color-aware* (isto é, que se baseia no anterior valor da probabilidade de perda além do volume de tráfego para efectuar a remarcação) do tráfego AF.

Na raíz da hierarquia da interface de egresso está instalada uma pseudo-disciplina de serviço `dsmark` que, ao contrário do que acontece no nó interior, não inicializa o campo `skb->tc_index`, uma vez que essa inicialização já foi feita na interface de ingresso. Outra diferença em relação ao nó interior é que aqui estão configuradas as pseudo-classes de `dsmark` para efectuar a (re)marcação do campo DSCP à saída.

Nesta instância de `dsmark` estão instalados filtros que seleccionam a classe a que cada pacote deve ser atribuído. De particular interesse são os que seleccionam o tráfego AF e que efectuam a (re)marcação *color-aware*. Assim, os elementos com preferência 1 seleccionam apenas o tráfego destinado aos PHB AFx1. Os blocos de policiamento remetem para posterior classificação o tráfego excedente. Os elementos com preferência 2 seleccionam todo o tráfego destinado a AFx e cujo *bit* de ordem 8 está a zero, ou seja, excluindo o tráfego de prioridade de perda 3 seleccionam apenas o excedente AFx1 e o tráfego AFx2. Este tráfego é novamente policiado, sendo o excedente posteriormente repescado pelo filtro de preferência 3, que usa apenas a classe AF para seleccionar os pacotes. Assim, é repescado o excedente de tráfego AFx1 e AFx2, juntamente com o tráfego AFx3, cujo débito conjunto é novamente policiado, sendo eliminado o tráfego excedente.

As restantes partes da hierarquia de egresso bem como os filtros nela instalados são em tudo idênticos ao caso do nó interior. Note-se que pelo facto de se reservar uma capacidade de 50 kbit/s para tráfego EF não marcado previamente, a capacidade máxima para tráfego previamente marcado com o DSCP EF é reduzida deste mesmo valor em relação ao caso do nó interior. Além disso, o policiamento configurado limita o total de tráfego EF a um pouco menos do que a capacidade reservada (245 kbit/s em vez de 250 kbit/s), pelo que a capacidade máxima para tráfego pré-marcado EF é de 195 kbit/s.

Dada a semelhança do controlo de tráfego aqui efectuado com o caso do nó interior, interessa

essencialmente testar o comportamento dos fluxos que vão ser marcados no *router*, em particular no que se refere ao DSCP atribuído e ao policiamento.

O primeiro teste efectuado consiste na geração de dois fluxos de pacotes sem marcação prévia do campo DSCP, com características que os levam a ser classificados um como EF e o outro como AF2x (x é determinado, pacote a pacote, pelo policiamento). Não existe qualquer carga adicional no *router*. Os resultados deste primeiro teste são apresentados na tabela 5.17.

Fluxo seleccionado	Prob. AF	Tr. oferec. kbit/s	Débito kbit/s	Perdas %
EF	—	60	48.4	19.3%
AF2	1	120	29.2	18.9%
	2		29.3	
	3		38.8	

Tabela 5.17: Primeiro teste sem carga adicional

Pode observar-se que o débito útil do fluxo EF é limitado a um valor muito próximo do configurado para policiamento. De igual modo, o fluxo AF2 vê os seus pacotes divididos pelas três probabilidades de perda da forma indicada pelos níveis de policiamento. Os pacotes excedentes foram eliminados em ambos os casos. É importante que isto aconteça, particularmente no fluxo EF, pois o facto de ser policiado implica que não vai afectar negativamente o tráfego EF pré-marcado.

Para verificar se não existem perdas no fluxo EF se este não exceder a sua capacidade configurada foi efectuado um segundo teste, cujos resultados se apresentam na tabela 5.18. Confirma-se assim o funcionamento correcto dos blocos de policiamento.

Fluxo seleccionado	Prob. AF	Tr. oferec. kbit/s	Débito kbit/s	Perdas %
EF	—	48	48	0
AF2	1	120	29.2	20.0%
	2		29.3	
	3		37.6	

Tabela 5.18: Segundo teste sem carga adicional

Por fim, foi realizado um terceiro teste, com fluxos idênticos aos do segundo teste, e ainda uma certa carga nos vários grupos criada pelo gerador de tráfego de enchimento. O tráfego de enchimento foi emitido por 70 segundos, durante os quais foram activados e desactivados os fluxos de teste, com duração de 60 segundos. Os resultados apresentam-se na tabela 5.19.

Tal como era esperado, não se verificaram perdas em nenhum dos fluxos do grupo EF[20]. Quanto ao tráfego tipo melhor esforço, verificou-se um nível elevado de perdas, devido ao congestionamento.

Quanto aos fluxos AF, apenas se apresentam os resultados agregados das várias probabilidades de perda. A razão para isto é que uma vez que existe remarcação, pacotes originalmente marcados

[20]De facto, num outro teste em que o tráfego EF não marcado excedia a sua capacidade, verificou-se que o policiamento impedia que as perdas verificadas se propagassem ao tráfego EF pré-marcado.

	Fluxo	Prob. AF	Tr. oferec. kbit/s	Débito kbit/s	Perdas %
Não marcado	EF	—	48	48	0
	AF2	1	120	28.8	56.6%
		2		15.0	
		3		7.5	
Pré-marcado	EF	—	186	186	0
	AF1	1	42	124	1.16%
		2	42		
		3	42		
	AF2	1	42	79.7	36.6%
		2	42		
		3	42		
	AF3	1	42	124	1.29%
		2	42		
		3	42		
	AF4	1	42	124	1.36%
		2	42		
		3	42		
	BE	—	1800	991	44.7%

Tabela 5.19: Teste em carga

com uma dada probabilidade de perda podem ser remarcados para uma superior[21]. A excepção é o fluxo AF2 sem marcação prévia, pois a única marcação aplicável é a efectuada no *router*, para o qual foi usada a ferramenta `tcpdump` para permitir a separação baseada no DSCP. Verifica-se que o tráfego pré-marcado AF2 sofre um nível de perdas superior ao de qualquer outro grupo AF. Isto deve-se ao facto de além deste tráfego, o grupo AF2 ir transportar algum tráfego sem marcação prévia. O tráfego AF2 sem marcação prévia sofre um nível de perdas muito elevado: 56.6%. Destas perdas, 20% ocorrem no policiamento inicial, como se pode observar no teste sem carga, sendo as restantes devidas ao policiamento global do grupo AF2 e ao congestionamento. Isto mostra claramente a necessidade de uma boa gestão de recursos, além dos mecanismos de controlo de tráfego, para a obtenção de serviços com qualidade: sobreutilização conduz necessariamente a congestionamento. Por fim, convém referir que a soma dos débitos AF2 excede os 125 kbit/s que lhe estão reservados. No entanto, é preciso não esquecer que os débitos apresentados são débitos médios, e que os fluxos sem marcação prévia tiveram uma duração inferior aos restantes. Assim, se se tiver em conta que durante cerca de 10 segundos o tráfego pré-marcado AF2 ocupou (provavelmente) os mesmos 124 kbit/s que o dos restantes grupos AF, facilmente se chega à conclusão que nos 60 segundos em que os fluxos sem marcação prévia existiram, o seu débito médio foi de 72.3 kbit/s, que somado ao débito total do tráfego AF2 sem marcação prévia é aproximadamente igual aos 124 kbit/s, valor consistente com o policiamento e a reserva de recursos efectuada.

[21] E nunca para uma menor probabilidade de perda, uma vez que a remarcação é *color-aware*.

Capítulo 6

Conclusões

Concluída a apresentação do trabalho desenvolvido, é necessário proceder à sua avaliação restrospectiva, nomeadamente no que respeita aos objectivos iniciais e em que medida em que foram atingidos, às contribuições efectuadas e à sua utilidade. Neste breve capítulo é efectuada essa análise e são ainda apontadas possíveis direcções para o prosseguimento do trabalho realizado.

Em relação ao objectivo inicial de criar um ambiente de serviços diferenciados com base em *hardware* comum e *software* de código aberto, pode dizer-se que foi amplamente atingido, pese embora a referida limitação de desempenho inerente ao controlador ATM utilizado. Esta limitação, no entanto, está perfeitamente identificada, sendo facilmente eliminada pelo uso de um controlador ATM diferente ou mesmo de outra tecnologia de rede, uma vez que este ambiente não faz uso das capacidades de garantia de qualidade de serviço da tecnologia ATM, que apenas foi utilizada pela flexibilidade na criação de diferentes configurações lógicas com ligações de capacidade controlável. Assim, não se trata de uma limitação real da validade do trabalho realizado: pelo contrário, serve para reforçar a ideia de que para obter qualidade de serviço é preciso, antes de mais, garantir o correcto funcionamento da rede em todas as camadas protocolares. Dito de outra forma, a existência de uma rede "saudável" a todos os níveis é um pré-requisito essencial para o fornecimento de serviços com qualidade. O não cumprimento deste pré-requisito torna infrutíferos os esforços de quaisquer mecanismos de controlo de tráfego para a obtenção de qualidade de serviço numa rede de comutação de pacotes.

Quanto ao sistema de teste implementado, igualmente com base num computador pessoal e *software* de código aberto (embora modificado), terá sido uma contribuição bastante positiva, pois é inegável a utilidade de um tal sistema. Se é certo que existe, disponível comercialmente, *hardware* especializado que permite testar qualidade de serviço em redes, também é verdade que o preço de tal *hardware* é bastante elevado quando comparado com o de um vulgar PC, o que torna desejável a existência de alternativas.

Futuramente, o sistema de teste poderá ser melhorado em duas vertentes: por um lado através do aumento da diversidade de testes possíveis, passando quer pela implementação de ainda maior funcionalidade na principal ferramenta de teste utilizada (`rude`), quer pela inclusão de outras ferra-

mentas; por outro lado através de uma maior automatização dos testes efectuados e da simplificação dos procedimentos a usar na geração de testes, incluindo eventualmente a implementação de uma interface gráfica para configuração e apresentação de resultados dos testes.

Relativamente à documentação detalhada da configuração dos mecanismos de controlo de tráfego do sistema operativo Linux, apesar de não se incluir nos objectivos iniciais terá sido uma contribuição de grande importância, uma vez que a falta de documentação tem sido um dos principais factores limitativos ao uso em mais larga escala dos mecanismos de controlo de tráfego / QoS e, mais recentemente, de suporte ao modelo de serviços diferenciados do sistema operativo Linux. Posteriormente, a documentação elaborada no quarto capítulo será traduzida para Inglês e disponibilizada separadamente na Internet por forma a atingir a mais vasta audiência possível.

No que se refere ao estudo das tecnologias para implementação de qualidade de serviço, dada a experiência adquirida e tratando-se de um assunto com especial interesse para o autor, o trabalho realizado será prosseguido quer na vertente de modelos arquitectónicos, quer no domínio dos algoritmos de escalonamento de pacotes (disciplinas de serviço).

Em relação aos modelos arquitectónicos, estão em curso desenvolvimentos em relação ao modelo Internet de serviços diferenciados (*diffserv*) que merecem especial atenção. Em particular, existe neste momento em versão *draft* um documento[1] que virá a substituir [11] e um outro[2] referente a um novo PHB, designado *Virtual Wire*. O PHB *Virtual Wire* pode ser implementado em qualquer domínio *diffserv* que suporte o PHB EF, e permite emular uma linha dedicada entre dois nós-fronteira (*edge*) desse domínio. Estes desenvolvimentos serão acompanhados com atenção.

Quanto aos algoritmos de escalonamento de pacotes, trata-se de uma área onde objectivos diferenciados (e mesmo contraditórios) de diferentes tipos de tráfego têm limitado de alguma forma o desenvolvimento de uma disciplina de serviço que permita suportar todos eles em simultâneo e de forma eficiente num *router*. Embora um algoritmo como o CBQ represente uma boa aproximação ao problema, particularmente se for dada a devida atenção à configuração dos diversos parâmetros, trata-se de uma solução ainda imperfeita. Assim, o escalonamento de pacotes é, sem dúvida, uma área de grande interesse para futura investigação.

[1] draft-ietf-diffserv-efresolve-00.txt
[2] draft-ietf-diffserv-pdb-vw-00.txt

Apêndice A

Como melhorar a granularidade temporal no Linux

Com a configuração-padrão, o núcleo do sistema operativo Linux apresenta uma granularidade temporal que é, em geral, demasiado grosseira para mecanismos rígidos de QoS. Isto deve-se ao facto de, pelo menos em arquitecturas `x86`, o valor-padrão para a constante interna do núcleo HZ, que representa a frequência, em Hz, do temporizador (*timer*) interno é de apenas 100. Além de impor um limite inferior ao intervalo de tempo usado para escalonar um evento no futuro, por exemplo o retirar de um pacote de uma fila de espera para envio, HZ impõe também um limite à precisão na medição do tempo no código de QoS e controlo de tráfego. Isto porque neste código o valor de tempo é medido com base na variável interna do núcleo `jiffies`, incrementada a cada interrupção gerada pelo temporizador. Por outro lado, o próprio escalonamento de processos de utilizador é baseado neste temporizador, pelo que em condições normais, um processo que esteja a gerar um fluxo de tráfego usado para teste pode ser interrompido por intervalos de tempo múltiplos de 10ms[1], dando origem a fluxos com um *jitter* de transmissão elevado. Este *jitter a priori* pode, só por si, conduzir à não-conformidade do fluxo com uma especificação de tráfego apertada. Neste apêndice são abordadas formas de contornar estes três problemas.

A.1 Medição do tempo

No ficheiro `include/net/pkt_sched.h` da árvore do código-fonte do núcleo Linux existe uma constante que define a forma de medição do tempo pelo código de QoS e controlo de tráfego. Esta constante, `PSCHED_CLOCK_SOURCE`, pode ser modificada[2] antes da compilação do núcleo, podendo tomar um de três valores:

`PSCHED_GETTIMEOFDAY` indica que deve usar-se a função interna do núcleo `do_gettimeofday()`

[1] Num sistema pouco carregado e sem processos dependentes apenas da CPU (e não de entrada/saída) no entanto, não é comum haver muitos instantes de mais de 10ms em que o processo fique parado.

[2] O programa de configuração do núcleo não permite fazê-lo de forma automática, pelo que é necessário editar o ficheiro manualmente.

para medir o tempo. Esta função, apesar de medir o tempo de forma precisa, é extremamente lenta, e obriga à manutenção do tempo numa estrutura pouco natural em que os segundos e os microssegundos são armazenados em campos diferentes. Além disso, pode dar origem a problemas quando são introduzidos atrasos superiores a 2 segundos, o que pode acontecer em classes com débito muito baixo.

`PSCHED_JIFFIES` é a forma padrão de medir o tempo. Baseando-se na variável `jiffies`, a sua precisão está limitada à granularidade do temporizador. Uma vez que são usadas técnicas de integração do relógio, e dados outros problemas arquitectónicos, isto pode não ser em si muito grave. No entanto pode tornar o policiamento não-fiável, particularmente em ligações com elevado débito.

`PSCHED_CPU` é a forma mais eficiente de medir o tempo, pois baseia-se em contadores internos ao próprio microprocessador, mas apenas é suportada em algumas arquitecturas[3], e em alguns casos pode ser alterado pelo suporte APM (*Advanced Power Management*) do BIOS (*Basic Input-Output System*) da máquina. Em máquinas `x86` não-portáteis recentes, no entanto, é este o método de eleição.

A.2 Temporizador

Porque o Linux foi originalmente concebido para funcionar em `i386`, e pretende continuar a funcionar mesmo nestas máquinas, o valor usado para a constante HZ em arquitecturas `x86` é 100, pois valores superiores elevariam desnecessária e demasiadamente o *overhead* da rotina de atendimento de interrupção do temporizador. Em máquinas recentes, particularmente desde o Pentium, pode elevar-se este valor para 1024 sem qualquer problema com o *overhead*. Já que o valor-padrão para HZ na arquitectura `alpha` é de 1024, este valor de HZ é bem suportado por praticamente todos os controladores de dispositivo (*device drivers*) e programas de utilizador.

Apenas pelo facto de alterar esta constante, é possível melhorar cerca de 10 vezes a característica de *jitter* de transmissão, bem como multiplicar por cerca de 10 o débito máximo controlável por uma disciplina de serviço não conservadora de trabalho, como um *token-bucket*.

Tal como no caso anterior, HZ não é directamente modificável a partir do programa de configuração do núcleo de forma automática, pelo que é necessário editar manualmente o ficheiro `include/asm/param.h` na árvore do código-fonte do núcleo.

A.3 Escalonamento de processos

Estando dependente do temporizador, a granularidade no escalonamento de processos é melhorada pelo simples facto de se usar um valor mais elevado de HZ. No entanto é possível melhorar ainda mais

[3] `x86` que implementem o TSC (*TimeStamp Counter*), isto é, praticamente todos desde o Pentium, e `alpha`.

o desempenho de programas geradores de tráfego usando mecanismos disponibilizado pelo sistema operativo Linux para suportar processos com requisitos, embora não rígidos, de tempo-real.

O primeiro mecanismo, invocado através de uma chamada a `sched_setscheduler()`, permite utilizar um esquema de prioridade estrita em vez do escalonamento normal. Todos os processos com escalonamento normal usam o método `SCHED_OTHER`, que tem a prioridade fixa 0. Por seu lado, processos especiais que alterem o escalonamento para `SCHED_RR`, com uma prioridade normalmente entre 1 e 90, são corridos com prioridade absoluta sobre todos os de prioridade inferior[4] (incluindo todos os `SCHED_OTHER`), podendo interrompê-los a qualquer momento. Esta capacidade é extremamente importante para um processo ser capaz de gerar um fluxo de tráfego com baixo *jitter* de transmissão. Deve, no entanto, ser usada com cautela, pois pode bloquear todos os outros processos a correr no sistema.

O segundo mecanismo, invocado através de uma chamada a `mlock()`, impede que a memória utilizada pelo processo seja paginada para disco, evitando que o processo fique parado pelo tempo necessário a ler a informação do disco e repô-la em memória RAM. Este mecanismo é normalmente usado em conjunção com o anterior.

Por serem extremamente perigosos (o primeiro pode consumir todo o tempo de CPU e o segundo quase todo o espaço de memória RAM), estes mecanismos apenas são acessíveis para o utilizador `root`. São, no entanto, importantes num programa que gere fluxos de tráfego para avaliação de mecanismos de controlo de tráfego e QoS.

[4] Se houver mais que um com a mesma prioridade, correm alternadamente.

Apêndice B

Patch ao programa de geração e recepção de tráfego

```
--- rude-0.50-orig/DOC/README.crude Mon Jan 24 08:41:03 2000
+++ rude-0.50-mod/DOC/README.crude Sat Jan 27 03:46:58 2001
@@ -62,4 +62,11 @@
  default vaulue is 0, which means until
  interrupted with CTRL+C.

--_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
\ No newline at end of file
+   -s #[,#...]    Instead of logging results to a file
+           calculate some statistics on-the-fly.
+
+   -P <priority>     Set the process real-time priority.
+           Only root can do that. Be carefull -
+           this might lock up your computer !!!
+
+-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
--- rude-0.50-orig/DOC/README.rude Mon Jan 24 08:35:40 2000
+++ rude-0.50-mod/DOC/README.rude Sat Jan 27 03:49:49 2001
@@ -34,7 +34,7 @@
  option is compulsory if you want to generate
  traffic.

--p <priority>Set the process real-time priority.
+ -P <priority>Set the process real-time priority.
  Only root can do that. Be carefull -
  this might lock up your computer !!!

@@ -127,6 +127,14 @@

 ----------------------------------------------------------------------------
```

```
+ TOS <id> <value>
+
+   * This optional command allows for the setting of the TOS byte of the
+     IP header. Some values may only be used by root. If rude fails to
+     set the specified TOS it uses the default instead.
+
+------------------------------------------------------------------------------
+
  <mtime> <id> MODIFY <type> [type parameters]

    * This command is valid only for the following flow type(s):
@@ -180,4 +188,4 @@
    last longer than the trace file describes, rude will start from the
    beginning (i.e. loops until STOPped).

--_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
\ No newline at end of file
+-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
--- rude-0.50-orig/DOC/example.cfg Mon Nov  8 09:35:22 1999
+++ rude-0.50-mod/DOC/example.cfg Sat Jan 27 03:51:54 2001
@@ -33,9 +33,12 @@
 ## Starts immediately at the specified START time with following params:
 ##   400 packets/second with 100 bytes/packet = 40kbytes/sec (1kbyte=1000bytes)
 ##
+## Sets the TOS for this flow to LOW_DELAY (0x10)
+##
 ## 9 seconds after that the flow is turned off...
 ##
 0000 0025 ON 3001 10.1.1.1:10001 CONSTANT 400 100
+TOS 0025 0x10
 9000 0025 OFF

 ## FLOW 3: (flow ID = 1)
--- rude-0.50-orig/crude/main.c Mon Jan 24 09:02:12 2000
+++ rude-0.50-mod/crude/main.c Thu Jan 25 17:59:33 2001
@@ -35,8 +35,37 @@
 #include <sys/stat.h>
 #include <sys/socket.h>
 #include <sys/time.h>
+#include <sched.h>
+#include <sys/mman.h>
 #include <arpa/inet.h>
 #include <netinet/in.h>
+#include <limits.h>
+
+
+/*
```

```
+ * Private struct for each flow of runtime statistics
+ */
+struct flow_stat {
+  unsigned long flowid;
+  unsigned long seqmin, seq;  /* Seq number of the first and current packets */
+  unsigned long rec;          /* Number of received packets */
+  unsigned long oos;          /* Number of packets out of sequence */
+  long long js_sec;           /* Jitter sum of seconds */
+  long long js_usec;          /* Jitter sum of microsseconds */
+  long long ds_sec;           /* Delay sum of seconds */
+  long long ds_usec;          /* Delay sum of microsseconds */
+  long last_tx_sec;
+  long last_tx_usec;
+  long last_rx_sec;
+  long last_rx_usec;
+  long last_delay_sec;
+  long last_delay_usec;
+  long first_rx_sec;
+  long first_rx_usec;
+  long max_jitter_sec;
+  long max_jitter_usec;
+  unsigned long long s_size;  /* Sum of all packet sizes */
+};


 /*
@@ -45,6 +74,8 @@
 static void usage(char *);
 static int  make_conn(unsigned short, char *);
 static int  decode_file(void);
+static int  runtime_stats(unsigned short, unsigned long);
+void        print_stats(void);
 static int  rec_to_file(unsigned short,unsigned long);
 static int  rec_n_print(unsigned short,unsigned long);
 void        crude_handler(int);
@@ -53,9 +84,11 @@
 /*
  * Global variables
  */
-int main_socket         = 0;  /* The socket to listen to              */
-int main_file           = 0;  /* File to read from/to write to        */
-unsigned long pkt_count = 0;  /* Couter for received/processed packets */
+int main_socket         = 0;     /* The socket to listen to                */
+int main_file           = 0;     /* File to read from/to write to          */
+unsigned long pkt_count = 0;     /* Counter for received/processed packets */
+struct flow_stat *flows = NULL;  /* List of flows for runtime statistics  */
+int nflows              = 0;
```

```
 int main(int argc, char **argv)
@@ -66,10 +99,16 @@
   char *ifipadd       = NULL;    /* pointer to interface IP address */
   unsigned short port = 10001;   /* default UDP port number         */
   long w_num          = 0;       /* # pkts to capture. 0=unlimited  */
+  int priority        = 0;
+  uid_t user_id       = getuid();
   int  cmd_char       = 0;
   int  retval         = 0;
   long temp1          = 0;
+  int ps_flag         = 0;
   struct sigaction action;
+  struct sched_param p;
+  char *sptr, *eptr;
+  struct flow_stat *newflows;

   printf("crude version %s, Copyright (C) 1999 Juha Laine and Sampo Saaristo\n"
   "crude comes with ABSOLUTELY NO WARRANTY!\n"
@@ -77,7 +116,7 @@
   "under GNU GENERAL PUBLIC LICENSE Version 2.\n",VERSION);

   while((retval >= 0) &&
-((cmd_char = getopt(argc,argv,"hvd:p:i:l:n:")) != EOF)){
+ ((cmd_char = getopt(argc,argv,"hvd:p:i:l:P:n:s:")) != EOF)){
     switch(cmd_char){
     case 'v':
       if((optind == 2) && (argc == 2)){
@@ -158,6 +197,23 @@
       }
       break;

+    case 'P':
+     if(optarg != NULL){
+   priority = atoi(optarg);
+   if((priority < 1) || (priority > 90)){
+      fprintf(stderr,"crude: priority must be between 1 to 90\n");
+      retval = -2;
+   }
+   if(user_id != 0){
+      fprintf(stderr,"crude: must be root to set the priority level\n");
+      retval = -2;
+   }
+      } else {
+   RUDEBUG1("crude: invalid commandline arguments!\n");
+   retval = -2;
+   }
+   break;
```

```
+
     case 'n':
       if(optarg != NULL){
  errno = 0;
@@ -177,6 +233,51 @@
       }
       break;

+    case 's':
+      if(optarg != NULL){
+   sptr = optarg;
+   do {
+     newflows = realloc(flows, (nflows + 1) * sizeof(struct flow_stat));
+     if (newflows == NULL) {
+       RUDEBUG1("crude: failed to allocate memory for statistics!\n");
+       retval = -6;
+       break;
+     }
+     flows = newflows;
+     flows[nflows].flowid = strtoul(sptr, &eptr, 10);
+     if (eptr == sptr || flows[nflows].flowid == ULONG_MAX) {
+       RUDEBUG1("crude: error reading flow IDs!\n");
+       retval = -6;
+       break;
+     }
+     if (*eptr && *eptr != ',') {
+       RUDEBUG1("crude: flow IDs must be separated by a comma!\n");
+       retval = -6;
+       break;
+     }
+     sptr = eptr + 1;
+     flows[nflows].rec = 0;
+     flows[nflows].oos = 0;
+     flows[nflows].seq = ULONG_MAX;
+     flows[nflows].seqmin = ULONG_MAX;
+     flows[nflows].js_sec = 0;
+     flows[nflows].js_usec = 0;
+     flows[nflows].ds_sec = 0;
+     flows[nflows].ds_usec = 0;
+     flows[nflows].s_size = 0;
+     flows[nflows].last_delay_sec = 0;
+     flows[nflows].last_delay_usec = 0;
+     flows[nflows].max_jitter_sec = 0;
+     flows[nflows].max_jitter_usec = 0;
+     /* The other fields are initialized when the first packet arrives */
+     nflows++;
+   } while (*eptr);
+      } else {
```

```
+ RUDEBUG1("crude: invalid commandline arguments!\n");
+ retval = -2;
+      }
+      break;
+
     default:
       usage(argv[0]);
       retval = -1;
@@ -187,6 +288,26 @@
   /* (if retval < 0 -> ERROR or/and EXIT IMMEDIATELY) */
   if(retval < 0){ goto crude_exit; }

+  /*
+   * If this process is owned by root we can do some tricks to
+   * improve the performance... (the -P option)
+   */
+  if((user_id == 0) && (priority > 0)){
+    /* Try to lock the memory to avoid paging delays */
+    if(mlockall(MCL_CURRENT | MCL_FUTURE) < 0){
+      RUDEBUG1("crude: memory lock failed: %s\n", strerror(errno));
+    }
+
+    /* Switch to Round-Robin-Real-Time Scheduling */
+    p.sched_priority = priority;
+    if(sched_setscheduler(0, SCHED_RR, &p) < 0){
+      RUDEBUG1("crude: sched_setscheduler() failed: %s\n",strerror(errno));
+      retval = -1;
+      goto crude_exit;
+    }
+    RUDEBUG7("crude: program priority set to %d\n", p.sched_priority);
+  }
+
   /* Activate the signal handler(s) */
   memset(&action, 0, sizeof(struct sigaction));
   action.sa_handler = crude_handler;
@@ -203,14 +324,36 @@
       RUDEBUG1("crude: couldn't create socket!\n");
       retval = -8;
     } else {
-      if(main_file > 0){ retval = rec_to_file(port,w_num); }
+      if (nflows > 0){ retval = runtime_stats(port,w_num); }
+      else if(main_file > 0){ retval = rec_to_file(port,w_num); }
       else { retval = rec_n_print(port,w_num); }
     }
   }

  crude_exit:
+  if (retval >= 0 && nflows > 0) { ps_flag = 1; }
```

```
+  /*
+   * Restore the tweaked settings...
+   */
+  if((user_id == 0) && (priority > 0)){
+    /* Restore the program priority */
+    p.sched_priority = 0;
+    if(sched_setscheduler(0, SCHED_OTHER, &p) < 0){
+      RUDEBUG1("crude: program priority restoring failed: %s\n",
+               strerror(errno));
+      retval = -1;
+    } else {
+      RUDEBUG7("crude: program priority restored\n");
+    }
+
+    /* Release the locked memory */
+    munlockall();
+  }
+
+  if(ps_flag){ print_stats(); }
   if(main_file > 0){ close(main_file); }
   if(main_socket > 0){ close(main_socket); }
+  if(flows){free(flows); }
   exit(retval);
 }

@@ -222,15 +365,17 @@
 {
   printf("\nusage: %s -h | -v | -d file | "
   "[-p port] [-i addr] [-l file] [-n #]\n\n"
- "\t-h       = print (this) short help and usage information\n"
- "\t-v       = print the version number and exit\n"
- "\t-d file  = decode the file to stdout\n"
- "\t-p port  = port to listen to\n"
- "\t-i addr  = numeric IP addres for the interface to listen to\n"
- "\t-l file  = direct undecoded output to file (fastest method!)\n"
- "\t           default is to decode the output to stdout\n"
- "\t-n #     = exit automatically after # packets has been logged.\n"
- "\t           use CTRL+C to exit the program otherwise\n\n", name);
+  "\t-h           = print (this) short help and usage information\n"
+  "\t-v           = print the version number and exit\n"
+  "\t-d file      = decode the file to stdout\n"
+  "\t-p port      = port to listen to\n"
+  "\t-i addr      = numeric IP addres for the interface to listen to\n"
+  "\t-l file      = direct undecoded output to file (fastest method!)\n"
+  "\t               default is to decode the output to stdout\n"
+    "\t-P priority  = process realtime priority {1-90}\n\n"
+  "\t-s #[,# ...] = don't record: get runtime stats for specified flows\n"
+  "\t-n #         = exit automatically after # packets has been logged.\n"
```

```
+  "\t               use CTRL+C to exit the program otherwise\n\n", name);
 }


@@ -239,7 +384,20 @@
  */
 void crude_handler(int value)
 {
+  struct sched_param pri;
   RUDEBUG7("\ncrude: SIGNAL caught - exiting...\n");
+
+  /* Check & restore process priority */
+  if((getuid() == 0) && (sched_getscheduler(0) != SCHED_OTHER)){
+    pri.sched_priority = 0;
+    if(sched_setscheduler(0, SCHED_OTHER, &pri) < 0){
+      RUDEBUG1("crude_handler: crude priority failure: %s\n",strerror(errno));
+    } else {
+      RUDEBUG7("crude_handler: crude priority restored\n");
+    }
+  }
+
+  if(nflows > 0){ print_stats(); }
   if(main_file > 0){ close(main_file); }
   if(main_socket > 0){ close(main_socket); }
   RUDEBUG1("\ncrude: captured/processed %lu packets\n", pkt_count);
@@ -332,6 +490,197 @@
   free(buffer);
   RUDEBUG1("crude: captured/processed %lu packets\n", pkt_count);
   return 0;
+}
+
+
+/*
+ * runtime_stats() - don't record: gather statistics at runtime...
+ * This routine
+ */
+static int runtime_stats(unsigned short port, unsigned long limit)
+{
+  long               rec_bytes = 0;     /* Bytes read            */
+  int                wri_bytes = 0;     /* Bytes written         */
+  int                src_len = sizeof(struct sockaddr_in);
+  struct sockaddr_in src_addr;
+  struct timeval     time1;
+  char buffer[PMAXSIZE];
+  struct udp_data *data = (struct udp_data *) buffer;
+  int i;
+  struct flow_stat *fsp;
+  unsigned long seq;
```

```
+  long tx_s;
+  long tx_us;
+  unsigned long flowid;
+  long delay_s;
+  long delay_us;
+  long jitter_s;
+  long jitter_us;
+
+  /* Initialize some variables */
+  memset(buffer,0,PMAXSIZE);
+
+  while(1){
+    rec_bytes = recvfrom(main_socket, buffer, PMAXSIZE, 0,
+  (struct sockaddr *)&src_addr, &src_len);
+    if(rec_bytes <= 0){
+      RUDEBUG1("crude: error when receiving packet: %s\n",strerror(errno));
+    } else {
+      gettimeofday(&time1, NULL);
+      flowid = ntohl(data->flow_id);
+      for (i = 0; i < nflows; i++) {
+        if (flows[i].flowid == flowid)
+          break;
+      }
+      if (i == nflows) {
+        continue;
+      }
+      fsp = &(flows[i]);
+      /* BUG: the remaining code in the loop should really be atomic, but
+       * how do we guaratee that?
+       * Anyway, if you stop your sources before hitting CTRL-C this should
+       * be OK. */
+      seq = ntohl(data->sequence_number);
+      fsp->rec++;
+      tx_s = ntohl(data->tx_time_seconds);
+      tx_us = ntohl(data->tx_time_useconds);
+      delay_s = time1.tv_sec - tx_s;
+      delay_us = time1.tv_usec - tx_us;
+      if (seq <= fsp->seq) {
+        fsp->oos++;
+        if (seq < fsp->seqmin) {
+          /* Should only occur in case of reordering of the first packets */
+          fsp->seqmin = seq;
+          if (fsp->rec == 1) {
+          /* First packet: do the initialization.
+           * Here we use a trick to move the initialization conditional out
+           * of the normal execution path: we initialized seqmin with
+           * ULONG_MAX so that when we receive the first packet it's sequence
+           * number is always less than that.  :-)  */
```

```
+             fsp->oos--; /* The first packet obviously isn't out of sequence */
+             fsp->first_rx_sec = time1.tv_sec;
+             fsp->first_rx_usec = time1.tv_usec;
+             fsp->seq = seq;
+             goto update_last;
+          }
+        }
+      } else {
+         fsp->seq = seq;
+      }
+      fsp->ds_sec += delay_s;
+      fsp->ds_usec += delay_us;
+      jitter_s = delay_s - fsp->last_delay_sec;
+      jitter_us = delay_us - fsp->last_delay_usec;
+      /* We need the absolute value, so: */
+      if (jitter_s < 0) {
+        jitter_s = -jitter_s;
+        jitter_us = -jitter_us;
+      }  /* At this point, jitter_s >= 0 */
+      if (jitter_us < 0) {
+         if (jitter_s == 0) {
+            jitter_us = -jitter_us;
+         } else {
+            jitter_us += 1000000;
+            jitter_s--;
+         }
+      }  /* Now both jitter components are positive */
+      fsp->js_sec += jitter_s;
+      fsp->js_usec += jitter_us;
+      if (jitter_s > fsp->max_jitter_sec ||
+          jitter_s == fsp->max_jitter_sec && jitter_us > fsp->max_jitter_usec) {
+        fsp->max_jitter_sec = jitter_s;
+        fsp->max_jitter_usec = jitter_us;
+      }
+update_last:
+      fsp->last_tx_sec = tx_s;
+      fsp->last_tx_usec = tx_us;
+      fsp->last_rx_sec = time1.tv_sec;
+      fsp->last_rx_usec = time1.tv_usec;
+      fsp->last_delay_sec = delay_s;
+      fsp->last_delay_usec = delay_us;
+      fsp->s_size += rec_bytes;
+    }
+
+    pkt_count++;
+    /* Note that we only count packets from the flows we are gathering
+     * statistics for: the others don't matter in this case. */
+    if((limit != 0) && (pkt_count >= limit)){ break; }
```

```
+  } /* end of while */
+
+  RUDEBUG1("\ncrude: captured/processed %lu packets\n", pkt_count);
+  return 0;
+}
+
+
+/*
+ * print_stats() - print the statistics at the end of a runtime statistics
+ *                 gathering session
+ */
+void print_stats(void)
+{
+  int i;
+  struct flow_stat *fsp;
+  long long sec, usec;
+  double interval;
+
+  printf("\n"
+         "Runtime statistics results: \n"
+         "--------------------------- \n");
+  for (i = 0; i < nflows; i++) {
+    fsp = &flows[i];
+    printf("\nFlow_ID=%lu \n", fsp->flowid);
+    printf("Packets: received=%lu   out-of-seq=%lu   "
+           "lost(est)=%lu \n", fsp->rec, fsp->oos,
+           fsp->rec > 0 ? (fsp->seq - fsp->seqmin + 1 - fsp->rec) : 0);
+    printf("Total bytes received=%llu \n", fsp->s_size);
+    if (fsp->rec > 0) {
+       printf("Sequence numbers: first=%lu   last=%lu \n",
+              fsp->seqmin, fsp->seq);
+    }
+    if (fsp->rec < 2) {
+      printf("Can't calculate statistics: not enough packets received. \n");
+      continue;
+    }
+    sec = fsp->ds_sec / fsp->rec;
+    usec = (fsp->ds_usec + 1000000 * (fsp->ds_sec % fsp->rec)) / fsp->rec;
+    sec += usec / 1000000;
+    usec %= 1000000;
+    if (sec < 0) {
+      /* Possible with dessynchronized clocks */
+      if (usec > 0) {
+        /* usec must be positive so printf() won't put a minus sign
+         * after the decimal dot. */
+        usec = 1000000 - usec;
+        sec++;
+      } else {
```

```
+        /* usec must be positive so printf() won't put a minus sign
+         * after the decimal dot. */
+        usec = - usec;
+      }
+    } else {
+      if (usec < 0) {
+        usec += 1000000;
+        sec--;
+      }
+    }
+    printf("Delay: average=%lld.%06llu   ", sec, usec);
+    /* Both components of all jitter values are positive, thus the sum of
+     * them is also positive */
+    sec = fsp->js_sec / (fsp->rec - 1);
+    usec = (fsp->js_usec + 1000000 * (fsp->js_sec % (fsp->rec - 1))) /
+           (fsp->rec - 1);
+    sec += usec / 1000000;
+    usec %= 1000000;
+    printf("jitter=%llu.%06llu   seconds \n", sec, usec);
+    printf("Absolute maximum jitter=%ld.%06ld   seconds \n",
+           fsp->max_jitter_sec, fsp->max_jitter_usec);
+    sec = (long) fsp->last_rx_sec - (long) fsp->first_rx_sec;
+    usec = (long) fsp->last_rx_usec - (long) fsp->first_rx_usec;
+    interval = (double) sec + (double) usec / 1000000.0;
+    printf("Throughput=%g   Bps  (from first to last packet received) \n",
+           (double) fsp->s_size / interval);
+  }
+  printf("\n");
 }


--- rude-0.50-orig/include/rude.h Mon Jan 24 08:49:54 2000
+++ rude-0.50-mod/include/rude.h Sat Jan 27 02:12:20 2001
@@ -93,6 +93,8 @@
   int success;                          /* Internal counters */
   int sequence_nmbr;                    /*                   */

+  int tos;                              /* IP TOS byte if positive */
+
   union {
     f_type              ftype;
     struct cbr_params   cbr;
--- rude-0.50-orig/rude/main.c Fri Jan 21 14:29:27 2000
+++ rude-0.50-mod/rude/main.c Sat Jan 27 03:10:44 2001
@@ -95,7 +95,7 @@
     exit(1);
   }
```

```
-  while(((opt_ret=getopt(argc,argv,"s:p:hv")) != EOF) && (retval == 0)){
+  while(((opt_ret=getopt(argc,argv,"s:P:hv")) != EOF) && (retval == 0)){
     switch(opt_ret){
     case 'v':
       if((optind == 2) && (argc == 2)){
@@ -130,7 +130,7 @@
       }
       break;

-    case 'p':
+    case 'P':
       if(optarg != NULL){
  priority = atoi(optarg);
  if((priority < 1) || (priority > 90)){
@@ -200,7 +200,7 @@

   /*
    * If this process is owned by root we can do some tricks to
-   * improve the performance... (the -p option)
+   * improve the performance... (the -P option)
    */
   if((user_id == 0) && (priority > 0)){
     /* Try to lock the memory to avoid paging delays */
@@ -290,7 +290,7 @@
   "\t-h            = print (this) short help and usage information\n"
   "\t-v            = print the version number and exit\n"
   "\t-s scriptfile = path to the flow configuration file\n"
- "\t-p priority   = process realtime priority {1-90}\n\n",name);
+  "\t-P priority   = process realtime priority {1-90}\n\n",name);
 } /* usage() */


@@ -427,6 +427,7 @@
   struct flow_cfg *flow = head;
   int retval = 0;
   int flags  = 0;
+  unsigned char tos;

   memset(&our_addr, 0, sizeof(struct sockaddr));
   ((struct sockaddr_in *)&our_addr)->sin_family = AF_INET;
@@ -456,6 +457,14 @@
     } else if((fcntl(flow->send_sock, F_SETFL, flags | O_NONBLOCK)) < 0){
       RUDEBUG1("open_sockets() - SETFLAGS O_NONBLOCK error\n");
       retval--;
+    }
+
+    if(flow->tos > 0){
+      tos = (unsigned char) flow->tos;
```

```
+      if(setsockopt(flow->send_sock, SOL_IP, IP_TOS, &tos, sizeof(tos)) == -1){
+        RUDEBUG1("Can't set TOS for flow %ld: using default...\n",
+                 flow->flow_id);
+      }
     }

   socket_error:
--- rude-0.50-orig/rude/parse.c Fri Jan 21 14:12:24 2000
+++ rude-0.50-mod/rude/parse.c Sat Jan 27 03:26:14 2001
@@ -302,6 +302,7 @@
     return(-2);
   }
   memset(new,0,sizeof(struct flow_cfg));
+  new->tos = -1;  /* By default, don't set the TOS */

   /* Do sanity check to the given parameters */
   if((time < 0) || (sport < 1024) || ((typenum=check_type(type)) < 0) ||
@@ -551,6 +552,7 @@
   int  read_lines        = 0;
   int  start_set         = 0;
   long int h,m,s,time,id = 0;
+  int  tos               = 0;
   char buffer[1024],cmd[12];

   /* Read the file line by line and parse the commands */
@@ -595,6 +597,26 @@

     /* check if START is set, so we can proceed... */
     if(start_set != 0){
+      if(strncasecmp(buffer,"TOS",3) == 0){
+        if(2 != sscanf(buffer,"%*3s %ld %i",&id,&tos)){
+   errors--;
+   RUDEBUG1("read_cfg() - invalid TOS clause (line #=%d)\n",read_lines);
+   continue;
+        }
+        temp = find_flow_id(id);
+        if (temp == NULL){
+   errors--;
+   RUDEBUG1("read_cfg() - invalid TOS flow (line #=%d)\n",read_lines);
+        } else {
+          if (tos < 0 || tos > 0xFF){
+   errors--;
+   RUDEBUG1("read_cfg() - invalid TOS value (line #=%d)\n",read_lines);
+          } else {
+            temp->tos = tos;
+          }
+        }
+        continue;
```

```
+       }
        if((3 != sscanf(buffer,"%ld %ld %10s",&time,&id,cmd)) || (time < 0)){
  errors--;
  RUDEBUG1("read_cfg() - invalid CMD (line #=%d)\n",read_lines);
```

Apêndice C

Script exemplo do filtro u32

```
#!/bin/sh

DEV=atm3

case "$1" in
  start)
tc filter add dev $DEV parent 1: protocol ip pref 5 handle 1:: u32 divisor 256

tc filter add dev $DEV parent 1: protocol ip pref 5 u32 ht 800:: order 1 \
  match ip dst 192.168.225.0/24 classid 1:1
tc filter add dev $DEV parent 1: protocol ip pref 5 u32 ht 800:: order 2 \
  match u32 0 0 hashkey mask 0x00ff0000 at 8 offset mask 0x0f00 shift 6 at 0 \
  link 1::
tc filter add dev $DEV parent 1: protocol ip pref 5 u32 ht 800:: order 3 \
  match u32 0 0 classid 1:5

tc filter add dev $DEV parent 1: protocol ip pref 5 u32 ht 1:: order 1 \
  sample ip protocol 6 0xff match tcp dst 23 0xffff classid 1:2
tc filter add dev $DEV parent 1: protocol ip pref 5 u32 ht 1:: order 2 \
  sample ip protocol 6 0xff match tcp dst 80 0xffff classid 1:3

tc filter add dev $DEV parent 1: protocol ip pref 5 u32 ht 1:: order 1 \
  sample ip protocol 17 0xff match ip protocol 17 0xff classid 1:4
    ;;
  stop)
tc filter del dev $DEV parent 1: protocol ip pref 5
    ;;
  *)
echo usage: `basename $0` \(start\|stop\)
exit 1
    ;;
esac
```

Apêndice D

Script de configuração NAT

```
#!/bin/sh

ITF0=bond
RealIP0=194.117.24.96
FakeIP0=192.168.193.96
Router0=194.117.24.158

ITF1=atm2
RealIP1=192.168.129.96
FakeIP1=192.168.255.97
Router1=192.168.129.158

# Carregar os módulos necessários para efectuar NAT

modprobe ip_tables
modprobe iptable_nat


# Configurar NAT

# Fluxo ETH->ATM (ida)
iptables -t nat -A POSTROUTING -o $ITF0 -s $RealIP0 -d $FakeIP1 \
  -j SNAT --to $FakeIP0
iptables -t nat -A PREROUTING -i $ITF1 -s $FakeIP0 -d $FakeIP1 \
  -j DNAT --to $RealIP1

# Fluxo ATM->ETH (volta)
iptables -t nat -A POSTROUTING -o $ITF1 -s $RealIP1 -d $FakeIP0 \
  -j SNAT --to $FakeIP1
iptables -t nat -A PREROUTING -i $ITF0 -s $FakeIP1 -d $FakeIP0 \
  -j DNAT --to $RealIP0
```

```
# Proxy ARP
# Comentar estas linhas se os endereços fictícios pertencerem a sub-redes
# diferentes dos verdadeiros. Se for este o caso, o Router0 deve usar
# RealIP0 como salto seguinte para FakeIP0 e o Router1 deve usar RealIP1
# como salto seguinte para FakeIP1.

arp -i $ITF0 -Ds $FakeIP0 $ITF0 pub
#arp -i $ITF1 -Ds $FakeIP1 $ITF1 pub


# Rotas para os endereços fictícios

ip route add $FakeIP1/32 via $Router0
ip route add $FakeIP0/32 via $Router1
```

Apêndice E

Scripts de configuração dos testes

E.1 *Script* para configuração do teste à opção `bounded` do CBQ

```
#!/bin/sh

case "$1" in
  start)
# Qdiscs
tc qdisc add dev atm3 root handle 1: cbq bandwidth 2Mbit allot 9200 cell 16 \
  avpkt 64 mpu 48

# Classes
tc class add dev atm3 parent 1: classid 1:1 cbq bandwidth 2Mbit rate 200Kbit \
  avpkt 64 prio 2 allot 9200 bounded
tc class add dev atm3 parent 1:1 classid 1:11 cbq bandwidth 2Mbit rate 50Kbit \
  avpkt 64 prio 2 allot 9200
tc class add dev atm3 parent 1:1 classid 1:12 cbq bandwidth 2Mbit rate 150Kbit \
  avpkt 64 prio 2 allot 9200

# Filters
tc filter add dev atm3 parent 1: protocol ip prio 5 handle 1: u32 divisor 1
tc filter add dev atm3 parent 1: prio 5 u32 match ip tos 0x20 0xff flowid 1:11
tc filter add dev atm3 parent 1: prio 5 u32 match ip tos 0x00 0x00 flowid 1:12
;;

  stop)
tc filter del dev atm3 parent 1: prio 5
tc class del dev atm3 classid 1:11
tc class del dev atm3 classid 1:12
tc class del dev atm3 classid 1:1
tc qdisc del dev atm3 root
;;
```

```
  *)
echo "Usage: `basename "$0"` {start|stop}"

esac
```

E.2 *Script* para configuração do teste à opção `isolated` do CBQ

```
#!/bin/sh

case "$1" in
  start)
# Qdiscs
tc qdisc add dev atm3 root handle 1: cbq bandwidth 2000Kbit allot 9200 \
  cell 16 avpkt 750 mpu 48

# Classes
tc class add dev atm3 parent 1: classid 1:1 cbq bandwidth 2000Kbit \
  rate 500Kbit avpkt 750 prio 2 allot 9200 bounded
tc class add dev atm3 parent 1: classid 1:2 cbq bandwidth 2000Kbit \
  rate 1500Kbit avpkt 750 prio 2 allot 9200
tc class add dev atm3 parent 1:1 classid 1:11 cbq bandwidth 2000Kbit \
  rate 250Kbit avpkt 750 prio 2 allot 9200
tc class add dev atm3 parent 1:1 classid 1:12 cbq bandwidth 2000Kbit \
  rate 250Kbit avpkt 750 prio 2 allot 9200 isolated
tc class add dev atm3 parent 1:2 classid 1:21 cbq bandwidth 2000Kbit \
  rate 750Kbit avpkt 750 prio 2 allot 9200
tc class add dev atm3 parent 1:2 classid 1:22 cbq bandwidth 2000Kbit \
  rate 750Kbit avpkt 750 prio 2 allot 9200

# Filters
tc filter add dev atm3 parent 1: protocol ip prio 5 handle 1: u32 divisor 1
tc filter add dev atm3 parent 1: prio 5 u32 match ip tos 0x20 0xff flowid 1:11
tc filter add dev atm3 parent 1: prio 5 u32 match ip tos 0x40 0xff flowid 1:12
tc filter add dev atm3 parent 1: prio 5 u32 match ip tos 0x60 0xff flowid 1:21
tc filter add dev atm3 parent 1: prio 5 u32 match ip tos 0x80 0xff flowid 1:22
;;
  stop)
tc filter del dev atm3 parent 1: prio 5
tc class del dev atm3 classid 1:22
tc class del dev atm3 classid 1:21
tc class del dev atm3 classid 1:2
tc class del dev atm3 classid 1:12
tc class del dev atm3 classid 1:11
tc class del dev atm3 classid 1:1
tc qdisc del dev atm3 root
;;
```

```
  *)
echo "Usage: ‘basename "$0"‘ {start|stop}"

esac
```

Apêndice F

Scripts diffserv

F.1 Nó interior (*core*)

```
#!/bin/sh -x

DEV=atm3
LINKCAP=2000000
# Débitos
EFRATE=250000
AF1xRATE=125000
AF2xRATE=125000
AF3xRATE=125000
AF4xRATE=125000
# Tamanho das filas
EFLIMIT=7500
AF1ThMin=2KB
AF1ThMax=6KB
AF1LIMIT=10KB
AF2ThMin=2KB
AF2ThMax=6KB
AF2LIMIT=10KB
AF3ThMin=2KB
AF3ThMax=6KB
AF3LIMIT=10KB
AF4ThMin=2KB
AF4ThMax=6KB
AF4LIMIT=10KB
BELIMIT=20KB
# Probabilidades GRED de perda no grupo AF
AF11PROB=0.02
AF12PROB=0.04
AF13PROB=0.08
AF21PROB=0.02
```

```
AF22PROB=0.04
AF23PROB=0.08
AF31PROB=0.02
AF32PROB=0.04
AF33PROB=0.08
AF41PROB=0.02
AF42PROB=0.04
AF43PROB=0.08
# Burst permitido de pacotes de tamanho médio em AF
AF1BURST=4
AF2BURST=4
AF3BURST=4
AF4BURST=4
# Parâmetros RED para tráfego BE
BEThMin=4KB
BEThMax=15KB
BEPROB=0.04
BEBURST=6

AVPKT=1000
MTU=1500
L2HLEN=8
MPU=48


case "$1" in
  start)
MTU2=$[$MTU + $L2HLEN]
AFRATE=$[$AF1xRATE + $AF2xRATE + $AF3xRATE + $AF4xRATE]
BERATE=$[$LINKCAP - $EFRATE - $AFRATE]
if [ $BERATE -lt 0 ]; then
  echo Capacidade excessiva reservada para EF/AF
  exit 1
fi

# Hierarquia de qdiscs e classes
tc qdisc add dev $DEV handle 1:0 root dsmark indices 64 set_tc_index \
  default_index 1
tc qdisc add dev $DEV parent 1:0 handle 2:0 cbq bandwidth $LINKCAP \
  avpkt $AVPKT
# EF
tc class add dev $DEV parent 2:0 classid 2:5 cbq bandwidth $LINKCAP \
  rate $EFRATE avpkt $AVPKT allot $MTU2 mpu $MPU prio 1
ip link set dev $DEV mtu $MTU
tc qdisc add dev $DEV parent 2:5 handle 25:0 tbf limit $EFLIMIT \
  rate $EFRATE burst $MTU2 mtu $MTU2 mpu $MPU
# AF1
tc class add dev $DEV parent 2:0 classid 2:1 cbq bandwidth $LINKCAP \
```

```
  rate $AF1xRATE avpkt $AVPKT allot $MTU2 mpu $MPU prio 2
tc qdisc add dev $DEV parent 2:1 handle 21:0 gred setup DPs 3 default 2 grio
tc qdisc change dev $DEV parent 2:1 handle 21:0 gred limit $AF1LIMIT \
  min $AF1ThMin max $AF1ThMax avpkt $AVPKT burst $AF1BURST bandwidth $LINKCAP \
  DP 1 probability $AF11PROB prio 2
tc qdisc change dev $DEV parent 2:1 handle 21:0 gred limit $AF1LIMIT \
  min $AF1ThMin max $AF1ThMax avpkt $AVPKT burst $AF1BURST bandwidth $LINKCAP \
  DP 2 probability $AF12PROB prio 3
tc qdisc change dev $DEV parent 2:1 handle 21:0 gred limit $AF1LIMIT \
  min $AF1ThMin max $AF1ThMax avpkt $AVPKT burst $AF1BURST bandwidth $LINKCAP \
  DP 3 probability $AF13PROB prio 4
# AF2
tc class add dev $DEV parent 2:0 classid 2:2 cbq bandwidth $LINKCAP \
  rate $AF2xRATE avpkt $AVPKT allot $MTU2 mpu $MPU prio 2
tc qdisc add dev $DEV parent 2:2 handle 22:0 gred setup DPs 3 default 2 grio
tc qdisc change dev $DEV parent 2:2 handle 22:0 gred limit $AF2LIMIT \
  min $AF2ThMin max $AF2ThMax avpkt $AVPKT burst $AF2BURST bandwidth $LINKCAP \
  DP 1 probability $AF21PROB prio 2
tc qdisc change dev $DEV parent 2:2 handle 22:0 gred limit $AF2LIMIT \
  min $AF2ThMin max $AF2ThMax avpkt $AVPKT burst $AF2BURST bandwidth $LINKCAP \
  DP 2 probability $AF22PROB prio 3
tc qdisc change dev $DEV parent 2:2 handle 22:0 gred limit $AF2LIMIT \
  min $AF2ThMin max $AF2ThMax avpkt $AVPKT burst $AF2BURST bandwidth $LINKCAP \
  DP 3 probability $AF23PROB prio 4
# AF3
tc class add dev $DEV parent 2:0 classid 2:3 cbq bandwidth $LINKCAP \
  rate $AF3xRATE avpkt $AVPKT allot $MTU2 mpu $MPU prio 2
tc qdisc add dev $DEV parent 2:3 handle 23:0 gred setup DPs 3 default 2 grio
tc qdisc change dev $DEV parent 2:3 handle 23:0 gred limit $AF3LIMIT \
  min $AF3ThMin max $AF3ThMax avpkt $AVPKT burst $AF3BURST bandwidth $LINKCAP \
  DP 1 probability $AF31PROB prio 2
tc qdisc change dev $DEV parent 2:3 handle 23:0 gred limit $AF3LIMIT \
  min $AF3ThMin max $AF3ThMax avpkt $AVPKT burst $AF3BURST bandwidth $LINKCAP \
  DP 2 probability $AF32PROB prio 3
tc qdisc change dev $DEV parent 2:3 handle 23:0 gred limit $AF3LIMIT \
  min $AF3ThMin max $AF3ThMax avpkt $AVPKT burst $AF3BURST bandwidth $LINKCAP \
  DP 3 probability $AF33PROB prio 4
# AF4
tc class add dev $DEV parent 2:0 classid 2:4 cbq bandwidth $LINKCAP \
  rate $AF4xRATE avpkt $AVPKT allot $MTU2 mpu $MPU prio 2
tc qdisc add dev $DEV parent 2:4 handle 24:0 gred setup DPs 3 default 2 grio
tc qdisc change dev $DEV parent 2:4 handle 24:0 gred limit $AF4LIMIT \
  min $AF4ThMin max $AF4ThMax avpkt $AVPKT burst $AF4BURST bandwidth $LINKCAP \
  DP 1 probability $AF41PROB prio 2
tc qdisc change dev $DEV parent 2:4 handle 24:0 gred limit $AF4LIMIT \
  min $AF4ThMin max $AF4ThMax avpkt $AVPKT burst $AF4BURST bandwidth $LINKCAP \
  DP 2 probability $AF42PROB prio 3
tc qdisc change dev $DEV parent 2:4 handle 24:0 gred limit $AF4LIMIT \
```

```
  min $AF4ThMin max $AF4ThMax avpkt $AVPKT burst $AF4BURST bandwidth $LINKCAP \
  DP 3 probability $AF43PROB prio 4
# BE
tc class add dev $DEV parent 2:0 classid 2:6 cbq bandwidth $LINKCAP \
  rate $BERATE avpkt $AVPKT allot $MTU2 mpu $MPU prio 3 \
  split 2:0 defmap 0xffff
tc qdisc add dev $DEV parent 2:6 handle 26:0 red limit $BELIMIT min $BEThMin \
  max $BEThMax avpkt $AVPKT burst $BEBURST probability $BEPROB \
  bandwidth $LINKCAP

# Filtro para separar PHB e prob. de perda
tc filter add dev $DEV parent 1:0 protocol ip pref 1 tcindex \
  mask 0xfc shift 2 pass_on
# EF
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x2e tcindex \
  classid 1:50
# AF1x
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x0a tcindex \
  classid 1:11
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x0c tcindex \
  classid 1:12
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x0e tcindex \
  classid 1:13
# AF2x
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x12 tcindex \
  classid 1:21
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x14 tcindex \
  classid 1:22
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x16 tcindex \
  classid 1:23
# AF3x
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x1a tcindex \
  classid 1:31
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x1c tcindex \
  classid 1:32
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x1e tcindex \
  classid 1:33
# AF4x
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x22 tcindex \
  classid 1:41
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x24 tcindex \
  classid 1:42
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x26 tcindex \
  classid 1:43

# Filtro para atribuição à classe propriamente dita
tc filter add dev $DEV parent 2:0 protocol ip pref 1 tcindex \
  mask 0xf0 shift 4 pass_on
```

```
# EF
tc filter add dev $DEV parent 2:0 protocol ip pref 1 handle 0x5 tcindex \
  classid 2:5
# AF
tc filter add dev $DEV parent 2:0 protocol ip pref 1 handle 0x1 tcindex \
  classid 2:1
tc filter add dev $DEV parent 2:0 protocol ip pref 1 handle 0x2 tcindex \
  classid 2:2
tc filter add dev $DEV parent 2:0 protocol ip pref 1 handle 0x3 tcindex \
  classid 2:3
tc filter add dev $DEV parent 2:0 protocol ip pref 1 handle 0x4 tcindex \
  classid 2:4
# BE
tc filter add dev $DEV parent 2:0 protocol ip pref 1 handle 0x0 tcindex \
  classid 2:6
;;

  stop)
tc filter del dev $DEV parent 2:0 protocol ip pref 1
tc filter del dev $DEV parent 1:0 protocol ip pref 1
tc qdisc del dev $DEV parent 2:6
tc qdisc del dev $DEV parent 2:4
tc qdisc del dev $DEV parent 2:3
tc qdisc del dev $DEV parent 2:2
tc qdisc del dev $DEV parent 2:1
tc qdisc del dev $DEV parent 2:5
tc class del dev $DEV parent 2:0 classid 2:6
tc class del dev $DEV parent 2:0 classid 2:4
tc class del dev $DEV parent 2:0 classid 2:3
tc class del dev $DEV parent 2:0 classid 2:2
tc class del dev $DEV parent 2:0 classid 2:1
tc class del dev $DEV parent 2:0 classid 2:5
tc qdisc del dev $DEV parent 1:0
tc qdisc del dev $DEV root
;;

  *)
echo "Usage: `basename "$0"` {start|stop}"
;;
esac
```

F.2 Nó fronteira (*edge*)

```
#!/bin/sh -x
```

```
INDEV=eth0
DEV=atm3
LINKCAP=2000000
# Débitos
EFRATE=250000
AF1xRATE=125000
AF2xRATE=125000
AF3xRATE=125000
AF4xRATE=125000
# Tamanho das filas
EFLIMIT=7500
AF1ThMin=2KB
AF1ThMax=6KB
AF1LIMIT=10KB
AF2ThMin=2KB
AF2ThMax=6KB
AF2LIMIT=10KB
AF3ThMin=2KB
AF3ThMax=6KB
AF3LIMIT=10KB
AF4ThMin=2KB
AF4ThMax=6KB
AF4LIMIT=10KB
BELIMIT=20KB
# Probabilidades GRED de perda no grupo AF
AF11PROB=0.02
AF12PROB=0.04
AF13PROB=0.08
AF21PROB=0.02
AF22PROB=0.04
AF23PROB=0.08
AF31PROB=0.02
AF32PROB=0.04
AF33PROB=0.08
AF41PROB=0.02
AF42PROB=0.04
AF43PROB=0.08
# Burst permitido de pacotes de tamanho médio em AF
AF1BURST=4
AF2BURST=4
AF3BURST=4
AF4BURST=4
# Parâmetros RED para tráfego BE
BEThMin=4KB
BEThMax=15KB
BEPROB=0.04
BEBURST=6
```

```
AVPKT=1000
MTU=1500
L2HLEN=8
MPU=48

# Policiamento
EFPR=245000    # Um pouco inferior à capacidade configurada
EFPB=6KB
AF11PR=125000
AF12PR=150000
AF13PR=200000
AF1PB=8KB
AF21PR=125000
AF22PR=150000
AF23PR=200000
AF2PB=8KB
AF31PR=125000
AF32PR=150000
AF33PR=200000
AF3PB=8KB
AF41PR=125000
AF42PR=150000
AF43PR=200000
AF4PB=8KB


case "$1" in
  start)
MTU2=$[$MTU + $L2HLEN]
AFRATE=$[$AF1xRATE + $AF2xRATE + $AF3xRATE + $AF4xRATE]
BERATE=$[$LINKCAP - $EFRATE - $AFRATE]
if [ $BERATE -lt 0 ]; then
  echo Capacidade excessiva reservada para EF/AF
  exit 1
fi

# Hierarquia de qdiscs e classes
tc qdisc add dev $DEV handle 1:0 root dsmark indices 128 default_index 1
tc qdisc add dev $DEV parent 1:0 handle 2:0 cbq bandwidth $LINKCAP \
  avpkt $AVPKT
# EF
tc class add dev $DEV parent 2:0 classid 2:5 cbq bandwidth $LINKCAP \
  rate $EFRATE avpkt $AVPKT allot $MTU2 mpu $MPU prio 1
ip link set dev $DEV mtu $MTU
tc qdisc add dev $DEV parent 2:5 handle 25:0 tbf limit $EFLIMIT \
  rate $EFRATE burst $MTU2 mtu $MTU2 mpu $MPU
# AF1
tc class add dev $DEV parent 2:0 classid 2:1 cbq bandwidth $LINKCAP \
```

```
  rate $AF1xRATE avpkt $AVPKT allot $MTU2 mpu $MPU prio 2
tc qdisc add dev $DEV parent 2:1 handle 21:0 gred setup DPs 3 default 2 grio
tc qdisc change dev $DEV parent 2:1 handle 21:0 gred limit $AF1LIMIT \
  min $AF1ThMin max $AF1ThMax avpkt $AVPKT burst $AF1BURST bandwidth $LINKCAP \
  DP 1 probability $AF11PROB prio 2
tc qdisc change dev $DEV parent 2:1 handle 21:0 gred limit $AF1LIMIT \
  min $AF1ThMin max $AF1ThMax avpkt $AVPKT burst $AF1BURST bandwidth $LINKCAP \
  DP 2 probability $AF12PROB prio 3
tc qdisc change dev $DEV parent 2:1 handle 21:0 gred limit $AF1LIMIT \
  min $AF1ThMin max $AF1ThMax avpkt $AVPKT burst $AF1BURST bandwidth $LINKCAP \
  DP 3 probability $AF13PROB prio 4
# AF2
tc class add dev $DEV parent 2:0 classid 2:2 cbq bandwidth $LINKCAP \
  rate $AF2xRATE avpkt $AVPKT allot $MTU2 mpu $MPU prio 2
tc qdisc add dev $DEV parent 2:2 handle 22:0 gred setup DPs 3 default 2 grio
tc qdisc change dev $DEV parent 2:2 handle 22:0 gred limit $AF2LIMIT \
  min $AF2ThMin max $AF2ThMax avpkt $AVPKT burst $AF2BURST bandwidth $LINKCAP \
  DP 1 probability $AF21PROB prio 2
tc qdisc change dev $DEV parent 2:2 handle 22:0 gred limit $AF2LIMIT \
  min $AF2ThMin max $AF2ThMax avpkt $AVPKT burst $AF2BURST bandwidth $LINKCAP \
  DP 2 probability $AF22PROB prio 3
tc qdisc change dev $DEV parent 2:2 handle 22:0 gred limit $AF2LIMIT \
  min $AF2ThMin max $AF2ThMax avpkt $AVPKT burst $AF2BURST bandwidth $LINKCAP \
  DP 3 probability $AF23PROB prio 4
# AF3
tc class add dev $DEV parent 2:0 classid 2:3 cbq bandwidth $LINKCAP \
  rate $AF3xRATE avpkt $AVPKT allot $MTU2 mpu $MPU prio 2
tc qdisc add dev $DEV parent 2:3 handle 23:0 gred setup DPs 3 default 2 grio
tc qdisc change dev $DEV parent 2:3 handle 23:0 gred limit $AF3LIMIT \
  min $AF3ThMin max $AF3ThMax avpkt $AVPKT burst $AF3BURST bandwidth $LINKCAP \
  DP 1 probability $AF31PROB prio 2
tc qdisc change dev $DEV parent 2:3 handle 23:0 gred limit $AF3LIMIT \
  min $AF3ThMin max $AF3ThMax avpkt $AVPKT burst $AF3BURST bandwidth $LINKCAP \
  DP 2 probability $AF32PROB prio 3
tc qdisc change dev $DEV parent 2:3 handle 23:0 gred limit $AF3LIMIT \
  min $AF3ThMin max $AF3ThMax avpkt $AVPKT burst $AF3BURST bandwidth $LINKCAP \
  DP 3 probability $AF33PROB prio 4
# AF4
tc class add dev $DEV parent 2:0 classid 2:4 cbq bandwidth $LINKCAP \
  rate $AF4xRATE avpkt $AVPKT allot $MTU2 mpu $MPU prio 2
tc qdisc add dev $DEV parent 2:4 handle 24:0 gred setup DPs 3 default 2 grio
tc qdisc change dev $DEV parent 2:4 handle 24:0 gred limit $AF4LIMIT \
  min $AF4ThMin max $AF4ThMax avpkt $AVPKT burst $AF4BURST bandwidth $LINKCAP \
  DP 1 probability $AF41PROB prio 2
tc qdisc change dev $DEV parent 2:4 handle 24:0 gred limit $AF4LIMIT \
  min $AF4ThMin max $AF4ThMax avpkt $AVPKT burst $AF4BURST bandwidth $LINKCAP \
  DP 2 probability $AF42PROB prio 3
tc qdisc change dev $DEV parent 2:4 handle 24:0 gred limit $AF4LIMIT \
```

```
  min $AF4ThMin max $AF4ThMax avpkt $AVPKT burst $AF4BURST bandwidth $LINKCAP \
  DP 3 probability $AF43PROB prio 4
# BE
tc class add dev $DEV parent 2:0 classid 2:6 cbq bandwidth $LINKCAP \
  rate $BERATE avpkt $AVPKT allot $MTU2 mpu $MPU prio 3 \
  split 2:0 defmap 0xffff
tc qdisc add dev $DEV parent 2:6 handle 26:0 red limit $BELIMIT min $BEThMin \
  max $BEThMax avpkt $AVPKT burst $BEBURST probability $BEPROB \
  bandwidth $LINKCAP

# "classes" para (re)marcação
# EF
tc class change dev $DEV parent 1:0 classid 0x50 dsmark mask 0x03 value 0xb8
# AF1x
tc class change dev $DEV parent 1:0 classid 0x11 dsmark mask 0x03 value 0x28
tc class change dev $DEV parent 1:0 classid 0x12 dsmark mask 0x03 value 0x30
tc class change dev $DEV parent 1:0 classid 0x13 dsmark mask 0x03 value 0x38
# AF2x
tc class change dev $DEV parent 1:0 classid 0x21 dsmark mask 0x03 value 0x48
tc class change dev $DEV parent 1:0 classid 0x22 dsmark mask 0x03 value 0x50
tc class change dev $DEV parent 1:0 classid 0x23 dsmark mask 0x03 value 0x58
# AF3x
tc class change dev $DEV parent 1:0 classid 0x31 dsmark mask 0x03 value 0x68
tc class change dev $DEV parent 1:0 classid 0x32 dsmark mask 0x03 value 0x70
tc class change dev $DEV parent 1:0 classid 0x33 dsmark mask 0x03 value 0x78
# AF4x
tc class change dev $DEV parent 1:0 classid 0x41 dsmark mask 0x03 value 0x88
tc class change dev $DEV parent 1:0 classid 0x42 dsmark mask 0x03 value 0x90
tc class change dev $DEV parent 1:0 classid 0x43 dsmark mask 0x03 value 0x98

# Filtro para separar PHB e prob. de perda
tc filter add dev $DEV parent 1:0 protocol ip pref 1 tcindex \
  mask 0x3f shift 0 pass_on
# EF
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x14 tcindex \
  classid 1:50 police rate $EFPR burst $EFPB action drop
# AF11
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x05 tcindex \
  classid 1:11 police rate $AF11PR burst $AF1PB action continue
# AF21
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x09 tcindex \
  classid 1:21 police rate $AF21PR burst $AF2PB action continue
# AF31
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x0d tcindex \
  classid 1:31 police rate $AF31PR burst $AF3PB action continue
# AF41
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x11 tcindex \
  classid 1:41 police rate $AF41PR burst $AF4PB action continue
```

```
# BE
tc filter add dev $DEV parent 1:0 protocol ip pref 1 handle 0x01 tcindex \
  classid 1:1

# Filtro para repescagem de AFx1 e selecção de AFx2
tc filter add dev $DEV parent 1:0 protocol ip pref 2 tcindex \
  mask 0x011C shift 2 pass_on
# AF11 repescado e AF12
tc filter add dev $DEV parent 1:0 protocol ip pref 2 handle 0x0001 tcindex \
  classid 1:12 police rate $AF12PR burst $AF1PB action continue
# AF21 repescado e AF22
tc filter add dev $DEV parent 1:0 protocol ip pref 2 handle 0x0002 tcindex \
  classid 1:22 police rate $AF22PR burst $AF2PB action continue
# AF31 repescado e AF32
tc filter add dev $DEV parent 1:0 protocol ip pref 2 handle 0x0003 tcindex \
  classid 1:32 police rate $AF32PR burst $AF3PB action continue
# AF41 repescado e AF42
tc filter add dev $DEV parent 1:0 protocol ip pref 2 handle 0x0004 tcindex \
  classid 1:42 police rate $AF42PR burst $AF4PB action continue

# Filtro para repescagem de AFx1 e AFx2 e selecção de AFx3
tc filter add dev $DEV parent 1:0 protocol ip pref 3 tcindex \
  mask 0x1C shift 2 pass_on
# AF11 e AF12 repescados e AF13
tc filter add dev $DEV parent 1:0 protocol ip pref 3 handle 0x0001 tcindex \
  classid 1:13 police rate $AF13PR burst $AF1PB action drop
# AF21 e AF22 repescados e AF23
tc filter add dev $DEV parent 1:0 protocol ip pref 3 handle 0x0002 tcindex \
  classid 1:23 police rate $AF23PR burst $AF2PB action drop
# AF31 e AF32 repescados e AF33
tc filter add dev $DEV parent 1:0 protocol ip pref 3 handle 0x0003 tcindex \
  classid 1:33 police rate $AF33PR burst $AF3PB action drop
# AF41 e AF42 repescados e AF43
tc filter add dev $DEV parent 1:0 protocol ip pref 3 handle 0x0004 tcindex \
  classid 1:43 police rate $AF43PR burst $AF4PB action drop

# Filtro para atribuição à classe propriamente dita
tc filter add dev $DEV parent 2:0 protocol ip pref 1 tcindex \
  mask 0xf0 shift 4 pass_on
# EF
tc filter add dev $DEV parent 2:0 protocol ip pref 1 handle 0x5 tcindex \
  classid 2:5
# AF
tc filter add dev $DEV parent 2:0 protocol ip pref 1 handle 0x1 tcindex \
  classid 2:1
tc filter add dev $DEV parent 2:0 protocol ip pref 1 handle 0x2 tcindex \
  classid 2:2
tc filter add dev $DEV parent 2:0 protocol ip pref 1 handle 0x3 tcindex \
```

```
  classid 2:3
tc filter add dev $DEV parent 2:0 protocol ip pref 1 handle 0x4 tcindex \
  classid 2:4
# BE
tc filter add dev $DEV parent 2:0 protocol ip pref 1 handle 0x0 tcindex \
  classid 2:6


# Ingresso

# qdisc
tc qdisc add dev $INDEV ingress

# Os dois bits menos significaticos vão conter a prob. de perda
# Os três bits seguintes a classe AF, 5 para BE e 0 para AF
# Nas classes AF, o bit de ordem 8 fica a 1 para probs. 3

# Filtros para pacotes não marcados
tc filter add dev $INDEV parent ffff:0 protocol ip pref 1 handle 1:: \
  u32 divisor 1
tc filter add dev $INDEV parent ffff:0 protocol ip pref 1 handle 800::1 \
  u32 match ip nofrag offset mask 0x0f00 shift 6 at 0 link 1::
# Se IP de origem = 192.168.193.96/32 e porta de origem = 3025 => EF
tc filter add dev $INDEV parent ffff:0 protocol ip pref 1 handle 1::1 \
  u32 match ip src 192.168.193.96/32 match u16 0x0bd1 0xffff at nexthdr+0 \
  ht 1:: classid :14 police rate 50000 burst $MTU2 action drop
# Se IP de destino = 192.168.225.97/32 => AF2x
tc filter add dev $INDEV parent ffff:0 protocol ip pref 1 handle 800::2 \
  u32 match ip dst 192.168.225.97/32 classid :9 \
  police rate 30000 burst 2KB action continue
tc filter add dev $INDEV parent ffff:0 protocol ip pref 2 handle 801::1 \
  u32 match ip dst 192.168.225.97/32 classid :A \
  police rate 30000 burst 3KB action continue
tc filter add dev $INDEV parent ffff:0 protocol ip pref 3 handle 802::1 \
  u32 match ip dst 192.168.225.97/32 classid :10B \
  police rate 40000 burst 4KB action drop

# Filtros para pacotes marcados
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle 2:: \
  u32 divisor 256
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle ::1 \
  u32 ht 803:: match ip tos 0x00 0x00 hashkey mask 0x00fc0000 at 0 link 2::
# EF
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle 2::1 \
  u32 sample ip tos 0xb8 0xfc match ip tos 0xb8 0xfc ht 2:: classid :14
# AF1x
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle 2::1 \
  u32 sample ip tos 0x28 0xfc match ip tos 0x28 0xfc ht 2:: classid :5
```

```
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle 2::1 \
  u32 sample ip tos 0x30 0xfc match ip tos 0x30 0xfc ht 2:: classid :6
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle 2::1 \
  u32 sample ip tos 0x38 0xfc match ip tos 0x38 0xfc ht 2:: classid :107
# AF2x
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle 2::1 \
  u32 sample ip tos 0x48 0xfc match ip tos 0x48 0xfc ht 2:: classid :9
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle 2::1 \
  u32 sample ip tos 0x50 0xfc match ip tos 0x50 0xfc ht 2:: classid :A
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle 2::1 \
  u32 sample ip tos 0x58 0xfc match ip tos 0x58 0xfc ht 2:: classid :10B
# AF3x
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle 2::1 \
  u32 sample ip tos 0x68 0xfc match ip tos 0x68 0xfc ht 2:: classid :D
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle 2::1 \
  u32 sample ip tos 0x70 0xfc match ip tos 0x70 0xfc ht 2:: classid :E
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle 2::1 \
  u32 sample ip tos 0x78 0xfc match ip tos 0x78 0xfc ht 2:: classid :10F
# AF4x
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle 2::1 \
  u32 sample ip tos 0x88 0xfc match ip tos 0x88 0xfc ht 2:: classid :11
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle 2::1 \
  u32 sample ip tos 0x90 0xfc match ip tos 0x90 0xfc ht 2:: classid :12
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle 2::1 \
  u32 sample ip tos 0x98 0xfc match ip tos 0x98 0xfc ht 2:: classid :113
# BE
tc filter add dev $INDEV parent ffff:0 protocol ip pref 10 handle 803::2 \
  u32 match ip tos 0x00 0x00 classid :1
;;

  stop)
tc filter del dev $INDEV parent ffff:0 protocol ip pref 10
tc filter del dev $INDEV parent ffff:0 protocol ip pref 3
tc filter del dev $INDEV parent ffff:0 protocol ip pref 2
tc filter del dev $INDEV parent ffff:0 protocol ip pref 1
tc qdisc del dev $INDEV ingress
tc filter del dev $DEV parent 2:0 protocol ip pref 1
tc filter del dev $DEV parent 1:0 protocol ip pref 1
tc qdisc del dev $DEV parent 2:6
tc qdisc del dev $DEV parent 2:4
tc qdisc del dev $DEV parent 2:3
tc qdisc del dev $DEV parent 2:2
tc qdisc del dev $DEV parent 2:1
tc qdisc del dev $DEV parent 2:5
tc class del dev $DEV parent 2:0 classid 2:6
tc class del dev $DEV parent 2:0 classid 2:4
tc class del dev $DEV parent 2:0 classid 2:3
tc class del dev $DEV parent 2:0 classid 2:2
```

```
tc class del dev $DEV parent 2:0 classid 2:1
tc class del dev $DEV parent 2:0 classid 2:5
tc qdisc del dev $DEV parent 1:0
tc qdisc del dev $DEV root
;;

  *)
echo "Usage: ‘basename "$0"‘ {start|stop}"
;;
esac
```

Bibliografia

[1] "ITU-T Recommendation I.321, B-ISDN Protocol reference model and it's application," 1991.

[2] J. Wroclawski, "Specification of the Controlled-Load Network Element Service," RFC 2211, MIT LCS, Set. 1997.

[3] S. Shenker, C. Partridge, e R. Guerin, "Specification of Guaranteed Quality of Service," RFC 2212, Xerox, BBN, IBM, Set. 1997.

[4] P. P. White e J. Crowcroft, "The Integrated Services in the Internet: State of the Art," em *Proc. IEEE, Vol. 85, Nº12*, pp. 1934–1946, Dez. 1997.

[5] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, e A. Sastry, "The COPS (Common Open Policy Service) Protocol," RFC 2748, Intel, Level 3, Cisco, IPHighway, AT&T, Jan. 2000.

[6] S. Herzog, J. Boyle, R. Cohen, D. Durham, R. Rajan, e A. Sastry, "COPS usage for RSVP," RFC 2749, IPHighway, Level 13, Cisco, Intel, AT&T, Jan. 2000.

[7] K. Nichols, S. Blake, F. Baker, e D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," RFC 2474, Cisco Systems, Torrent Networking Technologies, EMC Corporation, Dez. 1998.

[8] J. Postel, "Internet Protocol," RFC 791, Information Sciences Institute — University of Southern California, Set. 1981.

[9] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, e W. Weiss, "An Architecture for Differentiated Services," RFC 2475, Torrent Networking Technologies, EMC Corporation, Sun Microsystems, Nortel UK, Bell Labs, Lucent Technologies, Dez. 1998.

[10] J. Heinanen, F. Baker, W. Weiss, e J. Wroclawski, "Assured Forwarding PHB," RFC 2597, Telia Finland, Cisco Systems, Lucent Technologies, MIT LCS, Jun. 1999.

[11] V. Jacobson, K. Nichols, e K. Poduri, "An Expedited Forwarding PHB," RFC 2598, Cisco Systems, Bay Networks, Jun. 1999.

[12] A. Parekh, *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*. Tese de doutoramento, Massachusetts Institute of Technology, Out. 1992.

[13] J. C. R. Bennet e H. Zhang, "Hierarchical Packet Fair Queueing Algorithms," em *Proceedings of the ACM SIGCOMM´96*, pp. 143 – 156, Ago. 1996.

[14] D. D. Clark, S. Shenker, e L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," em *Proceedings of the ACM SIGCOMM'92*, pp. 14 – 26, 1992.

[15] S. Floyd e V. jacobson, "Link-sharing and Resource Management Models for Packet Networks," *IEEE/ACM Transactions on Networking*, vol. 3, Ago. 1995.

[16] S. Floyd, "Notes on Class-Based Queuing: Setting Paramters," Fev. 1996.

[17] P. McKenney, "Stochastic Fairness Queuing," em *Internetworking: Research and Experience, Vol. 2*, pp. 113 – 131, Jan. 1991.

[18] W. Almesberger, J. H. Salim, e A. Kuznetsov, "Differentiated Services on Linux," Jun. 1999.

[19] K. Ramakrishnan e S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP," RFC 2481, AT&T Labs Research, LBNL, Jan. 1999.

[20] Cisco Systems, http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/cc111/car.htm, *Commited Access Rate*.

[21] L. Berger e T. O'Malley, "RSVP Extensions for IPSEC Data Flows," RFC 2207, FORE Systems, BBN, Set. 1997.

[22] R. Atkinson, "IP Authentication Header," RFC 1826, Naval Research Laboratory, Ago. 1995.

[23] R. Atkinson, "IP Encapsulating Security Payload (ESP)," RFC 1827, Naval Research Laboratory, Ago. 1995.

[24] H. Schulzrinne, S. Casner, R. Frederick, e V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 1889, GMD Fokus, Precept Software Inc., Xerox Palo Alto Research Center, Lawrence Berkeley National Laboratory, Jan. 1996.