Towards a Middleware for Mobile Edge-Cloud Applications

João Rodrigues, Eduardo R. B. Marques, Luís M. B. Lopes and Fernando Silva
CRACS/INESC-TEC & Faculty of Science, University of Porto
e-mail: {joao.rodrigues,edrdo,lblopes,fds}@dcc.fc.up.pt

ABSTRACT

In the last decade, technological advances and improved manufacturing processes have significantly dropped the price tag of mobile devices such as smartphones and tablets whilst augmenting their storage and computational capabilities. Their ubiquity fostered research on mobile edge-clouds, formed by sets of such devices in close proximity, with the goal of mastering their global computational and storage resources. The development of crowd-sourcing applications that take advantage of such edge-clouds is, however, hampered by the complexity of network formation and maintenance, the intrinsic instability of wireless links and the heterogeneity of the hardware and operating systems in the devices. In this paper we present a middleware to deal with this complexity, providing a building block upon which crowd-sourcing applications may be built. We motivate the development of the middleware through a discussion of real-world applications, and present the middleware's architecture along with the associated components and current development status. The middleware takes form as a Java API for Android devices that allows for the establishment of links using heterogeneous communication technologies (e.g., Wifi-Direct, Bluetooth), and the combination of these links to form a logical edge-cloud network. On top of this functionality, services for edge computation, storage, and streaming are also being developed.

CCS CONCEPTS

• **Networks** → **Programming interfaces**; **Mobile networks**;

ACM Reference Format:
João Rodrigues, Eduardo R. B. Marques, Luís M. B. Lopes and Fernando Silva . 2017. Towards a Middleware for Mobile Edge-Cloud Applications. In *Proceedings of MECC'17: Middleware for Edge Clouds & Cloudlets, Las Vegas, NV, USA, December 11–15, 2017 (MECC'17),* 6 pages.
https://doi.org/10.1145/3152360.3152361

1 INTRODUCTION

Since their introduction, mobile devices such as smartphones and tablets have traditionally been seen as thin clients of network applications, depending on beefy servers to perform most computationally intensive tasks [1]. Subsequently, the Mobile Cloud Computing (MCC) paradigm emerged [2], letting computationally intensive tasks and large data sets be offloaded to cloud computing infrastructures for execution and/or storage. The MCC approach, however, is inadequate for applications with low latency and/or high bandwidth requirements. This motivated the appearance of cloudlets [3], that handle requests from mobile devices at the edge of the network while serving as caches for heavyweight cloud computing infrastructures. Thus, cloudlets bring some of the computational and storage power of traditional cloud computing to local, less powerful servers, but improving quality of service.

Beyond MCC and cloudlets, the ongoing technological evolution makes the case for mobile edge-clouds [4], networks formed by nearby mobile devices where infrastructural support may be optional. Smartphones and tablets have become ubiquitous [5] and typically feature powerful multi-core processors, several gigabytes of storage space and multiple communication interfaces (e.g., Wifi, Wifi-Direct, Bluetooth, 3G/4G). As such, they pack considerable computational power and can communicate in device-to-device (D2D) mode with low latency and high bandwidth. A whole class of services and applications is made possible by crowd-sourcing the resources of these devices in such mobile edge-clouds [6–11].

Despite the growing interest in this area, and the fact that the market is relatively homogeneous, being dominated by Google's Android and Apple's iOS operating systems, relatively few crowd-sourcing applications have been proposed, either as research prototypes or commercial products. One of the reasons for this meagre output is, we believe, the lack of adequate middleware to support the development of such applications, allowing programmers to abstract away from the intricacies of device-to-device communication using multiple protocols, from building and maintaining a mesh network of devices, from moving and storing data between devices, from scheduling computations over the network, and other complex operations. A middleware capable of providing an API and core services with such functionality would go a long way to make application development more agile.

In this paper we present the architecture of one such middleware that is being developed in the context of the Hyrax project[1]. Section 2 describes the applications that motivated and provided the foundations for the middleware architecture presented in Section 3. Section 4 discusses related work. Section 5 concludes the paper with a discussion of open issues and future work.

2 MOTIVATIONAL APPLICATIONS

The Hyrax project defined as its main goal to explore game changing applications in the context of mobile edge-networks. Towards this goal, three application scenarios were selected, presenting progressive levels of untethered operation from traditional cloud infrastructures. These applications were built from scratch using the Android API for all operations, without resorting to rooting of

[1] http://hyrax.dcc.fc.up.pt

MECC'17, December 11–15, 2017, Las Vegas, NV, USA

ACM ISBN 978-1-4503-5171-3/17/12…$15.00
https://doi.org/10.1145/3152360.3152361

devices or any operation that might render them useless for general use. The study of the scenarios and the development of the applications provided us with the required experience to identify the common denominator at various levels of abstraction, towards the development of a middleware that factors out these common functionalities and provides a productive tool upon which developers can build on.

2.1 User-generated replays

The User-Generated Replays (UGR) application [11] allows users to capture and share videos in a crowded venue, e.g., during a sport event (Figure 1). Traditional applications rely on infrastructural communications for data dissemination, leading to traffic congestion, e.g., infrastructural Wifi and 3G/4G access is often quite slow and limited in sporting venues with an attendance of thousands of people. UGR mitigates this problem by leveraging edge-cloud networks formed by nearby devices. Each individual device is able to generate videos and cache/disseminate from/to nearby devices, and uses infrastructural communications as a last resort, thus alleviating the load on the latter. In a real-world experiment, the use of UGR was able to offload congestion from infrastructural Wifi and 4G by as much as 80% in terms of connected users and close to 60% for video downloads, while improving video download speed by a factor of 3 for devices connected to the edge cloud.

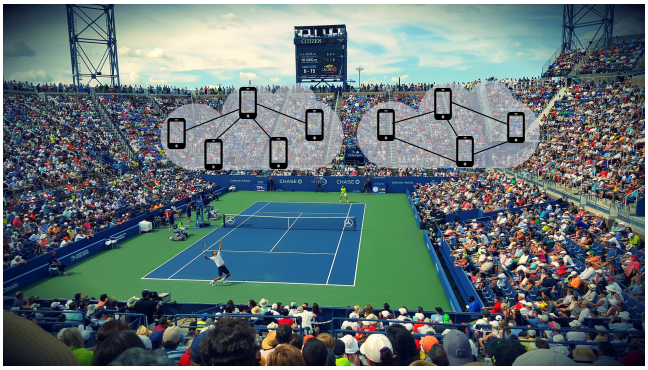

Figure 1: Video dissemination in large venues.

2.2 Distributed face recognition

We are developing a crowd-sourcing application for distributed face recognition to deal with emergencies like Amber alerts, used to disseminate information about missing people. We consider crowded scenarios such as stadiums or shopping malls, where it is common for instance to have children getting lost from their parents (Figure 2). By using computer vision on edge-clouds [12], it is possible to minimize, and potentially eliminate, the dependency on infrastructural clouds, by doing the search of the missing person's photo on the local storage of each mobile device without having to disclose them to a central cloud provider. The aim is to gather positive identifications of a missing person, along with important time and spatial information.

Figure 2: Searching for missing persons.

2.3 Rescue assistance in emergency scenarios

In recent years, several crowd-sourcing applications emerged to deal with emergency and disaster scenarios making use of mobile devices [13, 14]. Most of these applications, though, rely on standard communications and a web/cloud-based infrastructure, and are focused on mobile data collection and dissemination. In disaster scenarios, infrastructural communications (Wifi, GSM/ 3G/4G) may be severly impaired due to adverse environmental conditions, and people (e.g., a rescue team worker or a person in distress) find it hard or impossible to communicate (Figure 3). With this in mind, we are currently developing an application on top of Hyrax middleware that lets users find others that may be near the same physical vicinity, and establish peer-to-peer communication for the exchange of text messages or other media like audio/video, geo-tags, etc.

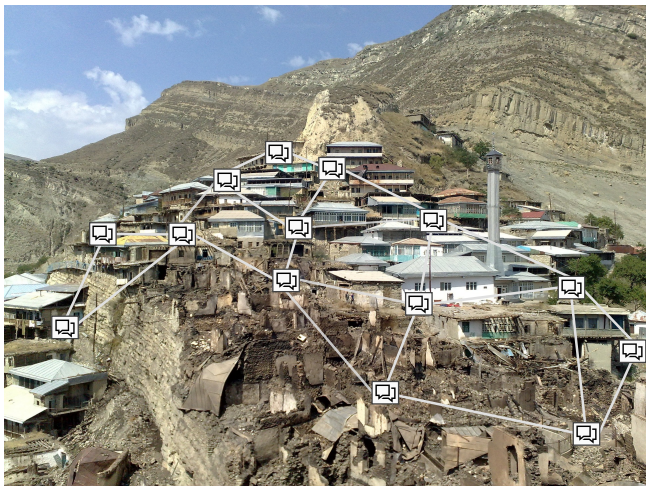

Figure 3: Communication in a disaster scenario.

3 THE HYRAX MIDDLEWARE

In this section we provide an overview of the Hyrax middleware. We first describe its architecture and provide a summary of the

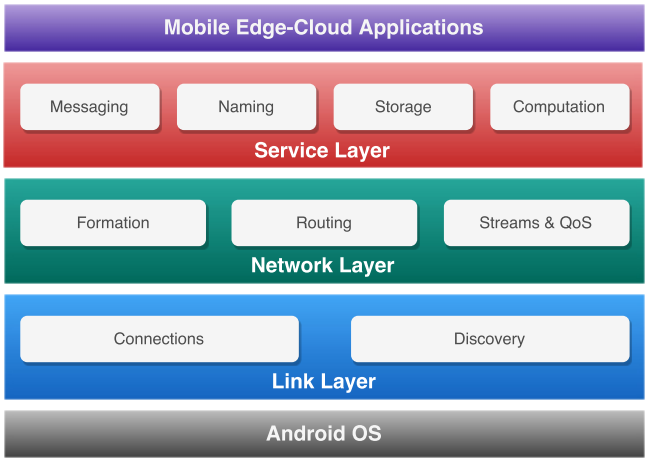

Figure 4: The Hyrax middleware architecture.

components in its layered structure. We then go through the middleware layers in more detail, describing their core functionality, design, and current implementation status.

3.1 Architecture

The Hyrax middleware architecture is illustrated in Figure 4. As shown, the middleware is used by mobile edge-cloud applications (shown at top) that run on top of the Android OS (bottom). The middleware comprises the following layers (bottom-up):

- The *Link Layer* is used to access the diverse communication technologies that may be available in a device, such as Wifi or Bluetooth. The primary concerns are to establish network links using these technologies and to provide the basic device discovery mechanisms required for proximity-awareness. For that purpose, it provides a normalised API interface that abstracts away the use of diverse technology-specific Android APIs.
- The *Network Layer* is responsible for defining and maintaining a logical network abstraction. It provides mechanisms for network formation and routing, that are automated, churn-tolerant, and employ diverse communication technologies simultaneously, forming an overlay network. On top of an up-and-running network, diverse types of communication, e.g., stream or packet-based, can be used with parametrisable levels of quality-of-service.
- The *Service Layer* comprises a set of core services that can be used directly by applications, such as for distributed storage (e.g., key-value stores), computation (e.g., for volunteer computing) or logical messaging (e.g., publish-subscribe). These services may offer a simple programming interface to applications, hiding away the complexity of the underlying network dynamics.

3.2 Link Layer

The organisation of the Link Layer is depicted in Figure 5. An API interface (shown top) is provided to deal with *links*; a link is an abstraction for a communication technology that is present on

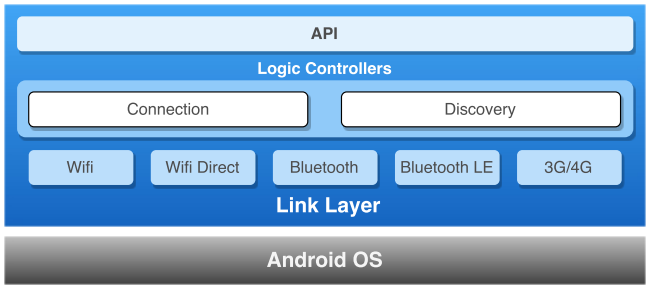

Figure 5: The link layer.

an Android device. Currently, the following types of link are supported (shown bottom in the figure): Wifi, Wifi-Direct, Bluetooth, Bluetooth Low Energy (LE) and 3G/4G. The API is a normalised interface to all of these, freeing the programmer from dealing with a set of heterogeneous, technology-specific Android APIs. The support for each technology in the Link Layer is organised as a set of logic controllers (shown middle), translating API operations into technology-specific commands. No root-level access ("rooted" device) is required, and the standard Android distributions are used.

The Link Layer operations concern device discovery and connection establishment. To give an example, the following sequence of calls sets up a device to become a Wifi-Direct access point (AP) and accepting connections from other devices:

```
Link link =
  LinkService.create(Technology.WIFI_DIRECT);
link.enable();
link.acceptConnections();
link.visible();
```

In the sequence, we first obtain a link handler to Wifi-Direct through the call to `LinkService.create()`, and make it active through the `enable()` operation. The link is then set up such that the device is visible by others, using `visible()`, and accepts connections (i.e., becomes an AP) through `acceptConnections()`). In symmetry, the following sequence of calls would discover all nearby devices and choose an AP to connect to:

```
Link link =
   LinkService.create(Technology.WIFI_DIRECT);
link.enable();
Outcome<Collection<Device>> o = link.discover();
if (o.isSuccessful()) {
  Collection<Device> devices = o.getResult();
  for (Device d : devices)
    if (someCondition(d)) link.connect(d);
}
```

As in the first sequence, we get a link handle and make it active. Thereafter, all devices that may be nearby are looked up using `discover()`. If a suitable access point exists, then it is chosen to form a Wifi-Direct connection through `connect()`. The illustrated code fragments would work similarly for other technologies merely by changing the link type argument, e.g., `Technology.BLUETOOTH` for Bluetooth.

3.3 Network Layer

The Network Layer is responsible for defining a logical network, more precisely an overlay network defined on top of communication links that can be enabled using the Link Layer. Thus, as illustrated by Figure 6, the Network Layer makes it possible to combine heterogeneous networks/communication technologies to yield the abstraction of a single logical network.

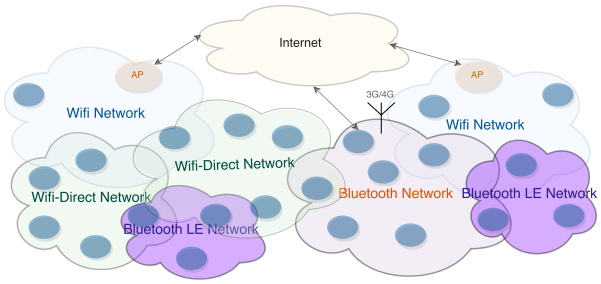

Figure 6: Logical edge-cloud network.

The organisation of the Network Layer is depicted in Figure 7. An API (top) defines operations for packet-based and stream-based communication, and, in addition, the enabling/disabling of physical network interfaces to use (bottom, at the Link Layer level). For instance, we may choose to define a network based solely on Bluetooth or combine Bluetooth and Wifi-Direct, and API hides any interaction with the Link Layer in the process.

Logical addresses are used to identify peers in the network in technology-agnostic manner (e.g., not tied to IP or MAC addresses), which are then translated on-the-fly to physical addresses when data is sent through actual network interfaces. Meta-information per device/logical address such as physical location or battery level, may guide the operation of routing algorithms which work on the logical address level. Under the hood, network formation algorithms guide the actual establishment of physical networks and their logical inter-connection.

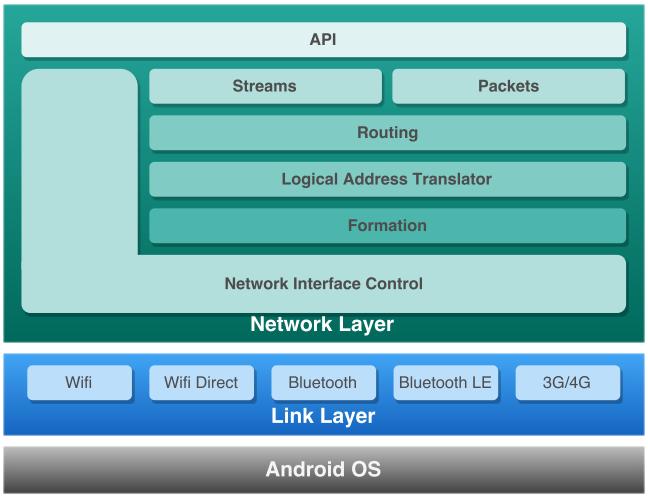

Figure 7: The Network Layer.

In the current version, we employ relatively simple algorithms for message routing and network formation. Routing is based on a scoped flooding strategy, something we expect to improve by leveraging/adapting well-established algorithms (e.g., AODV, OLSR) but possibly also geographical (position-based) routing [15]. Network formation works by defining star-like networks for each technology, which are inter-connected by bridge devices that participate in more than one such (sub-)network. For instance, a bridge device may be one that participates in a Bluetooth network and a Wifi-Direct network simultaneously, or one that is a Wifi-Direct access point and has access to infrastructural Wifi. Currently we are also implementing mesh formation algorithms, and considering support for Wifi-Direct multi-group formation [16].

We now illustrate the use of the API for simple packet-based data exchange. To bootstrap the network, we use `boot()` to obtain a reference to a `Network` object, specifying the (logical) routing algorithm to use, and enable the desired network interfaces through the `enableFormation()`. For instance, to start a network with flood-based routing, and enable Bluetooth and Wifi-Direct, we write:

```
Network net = NetworkService.boot(Routing.FLOOD)
net.enableFormation(Topology.BLUETOOTH_STAR)
net.enableFormation(Topology.WIFI_DIRECT_STAR)
```

To send messages over the network, `send()` routes messages to a specific device, and `sendToAll()` does so for all reachable devices:

```
Address specificPeer = ... ;
byte tag = MY_SERVICE_TAG;
byte[] data = ...;
net.send(tag, data, specificPeer);
net.sendToAll(tag, data);
```

The `sendToAll()` and `send()` operations can be combined meaningfully, e.g., `sendToAll()` may be used by a service running on a device to announce itself, and `send()` for peer-to-peer interaction after discovery. Note also that, as shown above, messages are sent with an associated (single-byte) tag. Tags facilitate message filtering by different services sitting on top of the network layer. To receive messages, a listener may be set up using `addMessageListener()`:

```
NetworkMessageListener listener = new
    NetworkMessageListener() {
  public void onMessage
  (byte tag, byte[] data, Address source) {
    // Handle incoming message
    ...
  }
};
net.addMessageListener(MY_SERVICE_TAG, listener);
```

3.4 Service Layer

On top of the Hyrax Network Layer, services can be deployed to provide common functionalities to edge-cloud applications. We

briefly refer to three such services developed in the scope of Hyrax project that are being ported to use the Hyrax middleware.

P^3-Mobile [17] is a service for opportunistic, best-effort parallel computing. The system builds a peer-to-peer hierarchical overlay on top of a Wifi or Wifi-Direct network. Computational tasks are subdivided for parallel execution as new devices join the overlay, but the scheme is also fault-tolerant when devices leave without notice. A snapshot mechanism keeps track of the completed portion of the workload for a given task. Thus, there is an on-the-fly reconfiguration of workload of the computation that is churn-tolerant. P^3-Mobile also implements a simple distributed key-value store that can be used as a communication channel between devices, e.g., for reporting the results of subtasks. In an early proof-of-concept experiment, substantial speedups were obtained for a simple benchmark application that used up to 16 Android devices.

Ephesus [18] is a distributed file storage service for mobile edge-clouds. It allows users to share files without requiring infrastructural communications, through a key-value store API implemented on top of a distributed hash table (DHT). To cope with churn, the service implements a best-effort approach to data consistency and persistence. The built-in mechanisms are adaptive to the popularity of data items, e.g., more frequently requested items are prioritised for replication among peers. Ephesus is also energy-aware, making a device disengage from the DHT when its battery level goes below a certain threshold. The system has been demonstrated by a shared photo gallery application (e.g., that can be used in party gatherings) where users take photos and share it with others.

Thyme is a time-aware publish-subscribe service for mobile edge-clouds under development, following up on the ideas presented in [15]. The service follows the typical operation of a publish-subscribe messaging system, but is time-aware in the sense that subscriptions have an associated time interval, i.e., are active only between specified start and end times. Geographical routing algorithms are employed, where devices are organised in spatial clusters, allowing messages to be routed without knowledge of the network topology or a priori route discovery. The point is to avoid the network communication overheads of standard routing protocols, and instead rely on the fact that device location has a strong connection to network topology in mobile edge-clouds.

4 RELATED WORK

4.1 Networking for edge-clouds

A number of APIs/toolkits have been developed in recent years in support of proximity-aware mobile applications that use peer-to-peer communications. In common with our middleware's network and link layers, the core concerns relate to the discovery of nearby devices and subsequent peer-to-peer communication, although in some cases relying on infrastructural support.

Apple's MultipeerConnectity framework for iOS [19] that can use infrastructural Wi-Fi, Wifi-Direct/Wi-Fi peer-to-peer and Bluetooth for device discovery and peer-to-peer communication. Infrastructural support is not mandatory for operation. Other commercial toolkits provide similar capabilities such as MeshKit from Opengarden [20] or P2Pkit from Ueppa [21].

Google Nearby [22] for Android and iOS combines Bluetooth, Bluetooth Low Energy, Wi-Fi, and near-ultrasonic audio for device discovery. However, the current version requires infrastructural (Internet and cloud) support to facilitate actual communication.

Alljoyn [23] is an open source framework for application discovery and communication that runs on several types of systems, including mobile Android devices. The Alljoyn API is based on network bus abstraction that hides the underlying networking dynamics. Under the hood, networking is implemented on top of Wi-Fi, Wi-Fi Direct and Powerline, but not Bluetooth.

WebRTC (Web Real-Time Communication) [24] designates a collection of open-source protocols and APIs that enable real-time communication over peer-to-peer connections. There is recent support for Android and iOS devices, although official support for strict peer-to-peer based discovery and communication is still unclear, e.g., an unofficial open-source project seeks to enable Wifi-Direct to work with WebRTC [25].

In spite of the availability of these systems, we decided to build a middleware from scratch based on three key factors. First, and foremost, to conduct research on mobile edge-clouds, we required full control of the software stack in order to gather reliable results. Thus we could not rely on closed-source systems. Second, systems that required infrastructure access to function could not be used as building blocks for our work either, as it involves scenarios of completely untethered computing. Third, we opted to develop for Android rather than iOS, as the Android ecosystem is more flexible regarding matters of customisation and software distribution. MeshKit, P2pKit, and MultipeerConnectity all provide some core functionalities that are close to those of our middleware, but are not open-source, and MultipeerConnectity works for iOS only. Google Nearby requires infrastructural support to pair devices, and Wi-Fi multicast in particular for multi-hop communication (unlike our middleware). Finally, Alljoyn and WebRTC require an underlying physical network already setup to function, and do not provide fine-grained programmatic control to setup and logically coordinate diverse communication technologies simultaneously.

4.2 Services and applications

On top of the base functionalities for device discovery and peer-to-peer communication, complementary services can be deployed in support of applications, but also already full-blown applications. For instance, some chat applications that do not require an Internet connection are quite popular for use in crowded venues, like ZombieChat [26] (that uses Apple's MultipeerConnectity) or FireChat [10] (based on Opengarden's Meshkit).

Streaming services and applications over mobile edge-clouds are also becoming significant. BitTorrent Live [27] allows peer-to-peer video streaming between mobile devices without an Internet connection. WebRTC, mentioned earlier, also deploys audio/video streaming services through the RTP protocol.

Distributed storage is another area of interest. Resilio Sync [28] is a popular application deployed on top of the BitTorrent protocol that allows file sharing over a private cloud enabled through Wifi-Direct. Research tools like iTrust [29], Krowd [6], along with Ephesus (Section 2), also implement distributed storage schemes for mobile edge-clouds.

In regard to distributed/parallel computing services using mobile devices like P^3-Mobile (Section 2), there were several research efforts, for instance: MMPI [7], for message-based parallel programs

written in a subset of MPI, and deployed over Bluetooth piconets; Honeybee [30] for task-based parallelism, using both Bluetooth and Wifi-Direct to form networks; the Hyrax [8] map-reduce engine for Android based on Hadoop; and FemtoClouds [31], a system for general computation offloading over a network of mobile devices.

5 DISCUSSION AND FUTURE WORK

We end the paper with a general summary of the current development status and future work. We then provide a separate discussion of security and privacy issues, that have not been addressed in this paper but are also crucial topics for future work.

5.1 Current status and evolutions

Currently, initial versions of the Hyrax middleware Link and Network Layers have been released[2] and are being integrated into services like P^3-mobile or applications like UGR (mentioned earlier in Section 2). Future work will concern evolutions of the Hyrax middleware in some key directions.

We plan to make the Link/Network layers more mature to incorporate configurable network formation and routing algorithms. We then wish to make a comparative evaluation of these algorithms using our applications in different settings (e.g., in relation to the churn level, network topology, mobility model). Integration with cloudlets is also a relevant line of work, enabling a broader range of applications and network dynamics.

Regarding applications for evaluation, beyond those already developed or being developed, we are also interested in data-intensive applications like torrent-based file sharing and video streaming, and crowd-sourcing sensing applications which tend to be less data-intensive but exhibit a high degree of mobility.

5.2 Security and privacy

Security and privacy are fundamental, cross-cutting issues for a middleware that supports crowd-sourcing applications.

In Hyrax we aim to address the basic building blocks needed to support a secure and distributed middleware, namely authorisation, authentication and auditing across heterogeneous trust domains. One problem is that most cryptographic primitives and security protocols rely on centralised trust entities, e.g., Public Key Infrastructure, thus they are not suitable for edge-clouds. We are currently conducting research on reputation-based mechanisms to detect malicious behaviours within a pure decentralised architecture, avoiding on centralised security oracles. Additional approaches are relevant, such as information flow control (IFC) [32] that assigns security tags for data flows within a distributed system. IFC suffers from tag explosion, however, that hampers its applicability in highly dynamically systems. Additionally, the secure management of IFC tags is still an open research topic.

On top of these mechanisms, additional schemes can be leveraged to enhance device security, including the use of memory enclaves, e.g., Android already employs ARM TrustZone, and the provision of other techniques for isolation/sandboxing. Complementary approaches such as proof-carrying code and homomorphic encryption provide strong guarantees, but are too demanding resource-wise for the mobile edge-cloud context.

[2]You may request access to these releases to the authors.

ACKNOWLEDGMENTS

This work is financed by ERDF (COMPETE 2020, POCI-01-0145-FEDER-006961), FCT (CMUP-ERI/FIA/-0048/2013 and UID/EEA/50014/2013), and FEDER (NORTE 2020, NORTE-01-0145-FEDER-000020).

REFERENCES

[1] The Internet Society. Internet Society Global Report 2015 - Mobile Evolution and Development of the Internet. https://www.internetsociety.org/globalinternetreport/2015/assets/download/IS_web.pdf, 2015.
[2] F. Liu, P. Shu, H. Jin, L. Ding, J. Yu, D. Niu, and B. Li. Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications. *IEEE Wireless Communications*, 20(3):14–22, 2013.
[3] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009.
[4] U. Drolia R. Martins, J. Tan, A. Chheda, M. Sanghavi, R. Gandhi, and P. Narasimhan. The case for mobile edge-clouds. In *Proc. UIC/ATC'13*. IEEE, 2013.
[5] P. Piejko. Global Mobile Statistics 2017. https://mobiforge.com/research-analysis/13-statistics-on-mobile-web-performance-in-2017.
[6] U. Drolia, N. Mickulicz, R. Gandhi, and P. Narasimhan. Krowd: A Key-Value Store for Crowded Venues. In *Proc. MobiArch'15*, pages 20–25. ACM, 2015.
[7] D. C. Doolan, S. Tabirca, and L. T. Yang. MMPI: a Message Passing Interface for the Mobile Environment. In *Proc. MoMM '08*, pages 317–321. ACM, 2008.
[8] E. E. Marinelli. Hyrax: Cloud Computing on Mobile Devices using MapReduce. Master's thesis, Master's Thesis, Carnegie Mellon University, 2009.
[9] T. Yan, M. Marzilli, R. Holmes, D. Ganesan, and M. Corner. mCrowd: A Platform for Mobile Crowdsourcing. In *Proc. SenSys'09*, pages 347–348. ACM, 2009.
[10] FireChat. http://opengarden.com/firechat/.
[11] P. M. P. Silva, J. Rodrigues, J. Silva, R. Martins, L. Lopes, and F. Silva. Using Edge-Clouds to Reduce Load on Traditional WiFi Infrastructure and Improve Quality of Experience. In *Proc. ICFEC'17*. IEEE Computer Society, 2017.
[12] E. Acosta, L. Torres, A. Albiol, and E. Delp. An Automatic Face Detection and Recognition System for Video Indexing Applications. In *Proc. ICASSP'02*, volume 4, pages IV–3644–IV–3647. IEEE, 2002.
[13] M. Poblet, E. García-Cuesta, and P. Casanovas. *Crowdsourcing Tools for Disaster Management: A Review of Platforms and Methods*, pages 261–274. Springer, 2014.
[14] F. Shih, O. Seneviratne, I. Liccardi, E. Patton, P. Meier, and C. Castillo. Democratizing mobile app development for disaster management. In *Proc. AIIP'13*, pages 39–42. ACM, 2013.
[15] J. A. Silva, J. Leitão, N. Preguiça, J. M. Lourenço, and H. Paulino. Towards the opportunistic combination of mobile ad-hoc networks with infrastructure access. In *Proc. MECC'16*, pages 3:1–3:6. ACM, 2016.
[16] A. Teófilo, D. Remédios, J. Lourenço, and H. Paulino. GOCRGO and GOGO: Two Minimal Communication Topologies for WiFi-Direct Multi-group Networking. In *Proc. MobiQuitous'17 (to appear)*. ACM, 2017.
[17] J. Silva, D. Silva, E. R. B. Marques, L. Lopes, and F. Silva. P3-Mobile: Parallel Computing for Mobile Edge-Clouds. In *Proc. CrossCloud'17*, pages 5:1–5:7. ACM, 2017.
[18] J. A. Silva, R. Monteiro, H. Paulino, and J. M. Lourenço. Ephemeral Data Storage for Networks of Hand-Held Devices. In *Proc. Trustcom/BigDataSE/ISPA*, pages 1106–1113. IEEE, 2016.
[19] Apple MultipeerConnectivity. https://developer.apple.com/reference/multipeerconnectivity.
[20] MeshKit SDK. https://www.opengarden.com/meshkit.html.
[21] Uepaa AG p2pkit. http://p2pkit.io/.
[22] Google Nearby. https://developers.google.com/nearby/messages/overview.
[23] Alljoyn Framework. https://allseenalliance.org/framework.
[24] WebRTC. https://webrtc.org/.
[25] WebRTC via WiFi Direct. https://github.com/jhkang/wifi-direct-webRTC.
[26] ZombieChat. http://getzombiechat.com/.
[27] BitTorrent Live. https://blive.tv.
[28] Resilio Sync. http://resilio.com.
[29] I. M. Lombera, L. E. Moser, P. M. Melliar-Smith, and Y.T. Chuang. Mobile ad-hoc search and retrieval in the iTrust over Wi-Fi Direct network. In *Proc. ICWMC'13*, pages 251–258. IARA, 2013.
[30] N. Fernando, S. W. Loke, and W. Rahayu. Honeybee: A Programming Framework for Mobile Crowd Computing. In *Proc. MobiQuitous'12*, pages 224–236. Springer, 2012.
[31] K. Habak, M. Ammar, K. A. Harras, and E. Zegura. Femto Clouds: Leveraging Mobile Devices to Provide Cloud Service at the Edge. In *Proc. CLOUD'15*, pages 9–16. IEEE, 2015.
[32] J. Singh, T. Pasquier, J. Bacon, J. Powles, R. Diaconu, and D. Eyers. Big Ideas Paper: Policy-driven Middleware for a Legally-compliant Internet of Things. In *Proc. Middleware'16*, pages 13:1–13:15. ACM, 2016.