UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXIX Ciclo

Peer-to-Peer Location-Based Services based on Blockchain and Web Technologies

Coordinatore:

Chiar.mo Prof. Marco Locatelli

Tutor:

Chiar.mo Prof. Francesco Zanichelli

Dottorando: *Giacomo Brambilla*

Dicembre 2016

“Well, here at last, dear friends, on the shores of the Sea
comes the end of our fellowship in Middle-earth. Go in peace!
I will not say: do not weep; for not all tears are an evil.”
J. R. R. Tolkien - The Return of the King

Summary

List of Figures

List of Tables

List of Listings

Introduction

In recent years, the interest in *smart cities* has been increasing remarkably. A smart city is an urban development vision that aims the integration of multiple information and communication technology (ICT) and Internet of Things (IoT) solutions in a secure fashion to manage city assets, such as local departments' information systems, schools, libraries, transportation systems, hospitals, power plants, water supply networks, waste management, law enforcement, and other community services. The goal of building a smart city is to improve quality of life by using *urban informatics* and technology to improve the efficiency of services and meet residents' needs.

ICT enables the enhancement of quality, performance and interactivity of urban services, the reduction of costs and resource consumption and the improvement of contact between citizens and government. Major technological, economic and environmental changes have generated interest in smart cities, including climate change, economic restructuring, the move to online retail and entertainment, ageing populations, urban population growth and pressures on public finances. For these reasons, the European Union (EU) has devoted constant efforts to devising a strategy for achieving *smart* urban growth for its metropolitan city-regions, with the development of a range of programmes under *Europe's Digital Agenda* [1] .

Under the smart city label, a myriad of technologies have been realized. That makes difficult to distil a precise description of a smart city; for that purpose, Deakin and Al Waer [2] listed four factors that contribute to its definition:

[1] *Digital Agenda for Europe* `https://ec.europa.eu/digital-single-market/`

1. the application of a wide range of electronic and digital technologies to communities and cities;
2. the use of ICT to transform life and working environments within the region;
3. the embedding of ICT in government systems;
4. the territorialisation of practices that brings ICT and people together to enhance the innovation and knowledge they offer.

According to Deakin, a smart city is a city that not only is provided with ICT in particular areas, but uses this technology to meet the demands of the market (the citizens of the city), in a manner that positively impacts the local community. In particular, a smart city uses information technologies to make more efficient use of physical infrastructures such as roads, buildings and other physical assets through artificial intelligence and data analytics. Moreover, ICT can effectively improve the engagement with citizens in local governance and decision, by use of open innovation processes and e-participation, improving the collective intelligence of the city's institutions through e-governance. Furthermore, smart cities can learn, adapt and innovate and thereby respond more effectively and promptly to changing circumstances by improving the intelligence of the city.

One envisioned distinctive feature of smart cities is the interconnection among mobile users and vehicles, to support the fulfillment of applications based on the geographical location of users and resources. This is one of the key aspects of smart cities since it enables anytime, anywhere access for citizens and visitors, promotes active participation, stimulates local commerce and the delivery of high relevant services. Especially, *location-based services* (LBSs) are information or entertainment services where the requests, the responses and served contents depend on the physical position of the requesting device [3].

LBSs include services to identify a location of a person or an object, such as discovering the nearest ATM, or parcel tracking and vehicle tracking services. Mobile

commerce, when taking the form of advertising directed at customers based on their current location, weather services and even location-based games are other examples of LBSs. Moreover, they can be used in a large variety of contexts, such as health, entertainment, work, personal life.

Nowadays, existing LBSs mainly rely on centralized architectures, managed by a few organizations that can afford the high performance computing systems required for such services, where there is an enormous number of requests that mobile users ask for, while continuously changing their location very quickly. Besides the fact that systems with a high degree of centralization hardly scale and servers can become bottlenecks or even single point of failures, it may not be desirable to relinquish the control of data over one single organization. Nevertheless, if on the one hand, large IT companies such as Google and Facebook are pushing more and more their LBSs without worrying too much about user privacy, on the other hand, researchers are investigating to provide such services while preserving user privacy [4].

In this context, various *peer-to-peer* (P2P) solutions have been proposed. These P2P protocols, in addition to safeguard privacy of users inasmuch the data are not in the hands of a single possibly untrustworthy company, support the realization of bottom-up LBSs, not requiring large and expensive infrastructures. Moreover, the decentralized nature of peer-to-peer networks increases the robustness, because it removes the single point of failure that can be inherent in a client-server based system [5], albeit, on the other hand, it increases the architectural complexity of the system itself, in particular from the perspective of routing and resource discovery. In fact, to design reliable services that take into account geographical locations while preserving the privacy of the users and supporting millions of connected devices, can be a very challenging and demanding task. Despite the many benefits of a P2P approach, often these solutions have been studied only in simulative environment and truly usable implementations have never been released.

Research Problem and Thesis Contribution

This Thesis aims at the definition and realization of a peer-to-peer overlay scheme where each peer can efficiently retrieve neighbors and resources located near any chosen geographical position, thus suitable for the fulfillment of location-based services. The idea is that the responsibility for maintaining information about position of active peers, *i.e.*, users participating in the LBS, are properly distributed in the peer-to-peer network, therefore a change in the set of participants causes only a minimal amount of disruption without reducing the quality of provided services.

As demonstrated in this Thesis, peer-to-peer is an appealing approach to design LBSs since, as already demonstrated for several types for distributed applications such as file sharing, multimedia streaming and content delivery solutions, as well as the innovative peer-to-peer-based digital cryptocurrencies, it is possible to manage a relevant number of concurrently active nodes, with a low cost but resilient solution.

In this Thesis, the designed peer-to-peer overlay has been intensively evaluated, both through simulations in various and different scenarios of mobility, and by means of field testing performed with the collaboration of several volunteers. In fact, the peer-to-peer overlay scheme has been effectively implemented as a software framework and used to realize a real peer-to-peer LBS. Furthermore, the problem related to truthfulness of geographical locations stated by peers has been faced, proposing a privacy-aware approach inspired by the novel *blockchain* technology.

The main contributions of this Thesis can be summarized as follows:

- An analysis on existing LBSs, having special consideration for those based on peer-to-peer overlay schemes. In particular, their main drawbacks have been identified and novel solutions to overcome them have been studied.

- The *Adaptive Distributed Geographic Table* (ADGT), a novel peer-to-peer overlay scheme for the realization of location-based services. The ADGT, in comparison with other peer-to-peer overlay schemes with similar purposes, ful-

fills all the requirements that are essential to LBSs, such as geographic broadcast or the retrieval of resources near any geographical location.

- A complete web application framework, *Adgt.js*, that is a truly cross-platform implementation of ADGT, released online with a free and open source software license and usable without restrictions to implement real LBSs.

- A novel Java-based simulation platform, which allows to easily simulate interconnected devices, by providing an intuitive simulation methodology, which results in maximal code reuse. The proposed platform provides a general-purpose simulation engine, which includes specific packages to simulate mobility, networking, and energy consumption models. Moreover, it allows to define general-purpose devices, which can be characterized by multiple network interfaces and protocols, as well as different network and energy models. With this solution, we can easily tackle problems like flexibility, modularity, code reuse and ease of deployment.

- An intensive evaluation of the ADGT through simulations in various mobility scenarios, measuring quantitative metrics that show the cost of the ADGT in terms of transmitted data, and qualitative metrics that show the behavior of the ADGT from a point of view of the quality of the service.

- The results of an activity of field testing with some volunteers that have collaborated evaluating a web application based on the Adgt.js framework. The realized application, in addition to experiment the versatility of Adgt.js, has allowed us to extrapolate the same indicators obtained in simulation, in order to compare the behavior of the system in simulation to the software implementation, really installed on mobile devices.

- An algorithm for distributed consensus among peers of the network that, with a novel and innovative mechanism inspired by the well-known blockchain of the popular Bitcoin protocol, allows to share and validate proofs-of-location, *i.e.* certificates that guarantee that the geographic location of a peer is actually true.

The proposed algorithm has been formally described and evaluated through simulations.

Thesis Outline

The Thesis is organized as follows:

- **Chapter 1**. The chapter describes and analyzes the two main approaches for the realization of LBSs available in literature. Furthermore, the chapter presents the current state of the art illustrating the most relevant and innovative LBS applications and research projects.

- **Chapter 2**. In this chapter, our novel peer-to-peer overlay scheme suitable for the development of LBSs, namely the Adaptive Distributed Geographic Table (ADGT), is described. It allows to efficiently retrieve peers or resources, to broadcast messages within any geographical region, and to be automatically notified about any type of information around any geographical location, following the publish/subscribe model

- **Chapter 3**. This chapter describes the implementation of the aforementioned georeferenced peer-to-peer overlay scheme. The ADGT has been realized exclusively with standard and open web technologies in order to obtain a truly cross-platform application framework.

- **Chapter 4**. In this chapter, it is described how the peer-to-peer overlay scheme has been extensively evaluated with OSMobility, which allows to simulate the motion of different entities in realistic geographical spaces and, adopting the software-in-the-loop simulation methodology, allows to test deployment software on simulated devices, immersed in simulated environments. The conducted performance analysis shows that the ADGT is cost-effective in terms of data rate, and therefore highly suited to mobile devices.

- **Chapter 5**. This chapter presents a concrete LBS example, based on Adgt.js, that clearly illustrates how straightforward and powerful such a framework is

and, by means of this, a performance evaluation of the framework in a real environment is presented.

- **Chapter 6**. This chapter illustrates a novel approach for producing proofs-of-location, *i.e.*, digital certificates that attest someone's presence at a certain geographic location, at some point in time, whereby LBSs can validate user locations. In particular, our approach relies on the blockchain — mostly known for being Bitcoin's core technology — to design a completely decentralized peer-to-peer architecture that guarantees location trustworthiness and preserves user privacy, at the same time. Simulation-based evaluation of the proposed technique shows an effective and robust behavior even in presence of significant shares of deceitful peers.

- **Chapter 7**. Finally, this chapter concludes the Thesis providing a general discussion and an outline of future work.

Chapter 1

Location-Based Services

Location-based services are gaining more and more importance for a broad range of applications, such as road/highway monitoring, emergency management, social networking and advertising. In this chapter, we analyze the most interesting approaches and algorithms, distinguishing between centralized and peer-to-peer architectures.

1.1 Centralized Architectures

Traditionally proposed architectures for location-based services are based on a centralized approach where one or more central servers have the responsibility to manage all position updates and queries from involved users, related for example to a specific point-of-interest, neighborhood discovery or path planning. In order to manage a huge number of active users at the same time, with a high quality of service, usually those solutions require a relevant computational power on the server side. For this reason, such services are typically provided by large companies that can afford the necessary powerful and expensive centralized systems, with severe consequences from the privacy viewpoint, since a large amount of data are controlled by a few organizations.

The Google Maps is a mobile app developed by Google, first released for Android on September 23, 2008, and later for iOS on December 13, 2012. Such an app is based on the web mapping service, developed by Google, and provides turn-by-turn

navigation, street view, public transit information, traffic view, search along routes, and many more features.

Figure 1.1: Screenshot of Google Maps running on iOS 7.

Google Maps' location tracking is widely regarded as a threat to user privacy [6]. Data on user life (personal habits and movements) are presumed to be used to make routing suggestions more precise.

Another example of centralized LBS is Foursquare, a mobile app which provides a local search-and-discovery service for its users. By taking into account the places where users go, the things they like, and what their friends advice, Foursquare provides recommendations of the places to visit in surroundings of users' current location. Differently from Google Maps, Foursquare is mainly oriented to be a local search and discovery tool, where users can "follow" other users to receive local recommendations from them. Foursquare provides functionalities to search for places of interest in surrounding area of users or by entering the name of a remote location, and

displays personalised recommendations based on the time of day, and on factors that include the users history, their venue ratings and according to their friends' reviews.

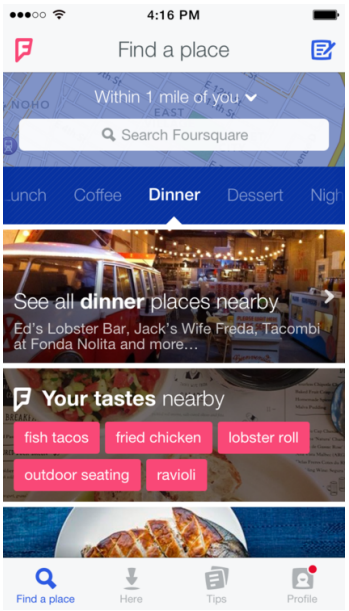


Figure 1.2: Foursquare showing personalized recommendations of places near the user's current location.

Privacy concerns on Foursquare raised when an ethical computer hacker discovered a vulnerability that allowed him to collect around 875000 "check-in" information, even if users chose to share these information only with their friends [7]. Moreover, on May 8, 2012 Foursquare developers announced a change to the API in response to a number of so-called "stalker" applications which had been making the locations of *e.g.* all female users within a specific area available to the public [8].

Pokémon Go is a free-to-play, location-based augmented reality game developed by Niantic for iOS, Android, and Apple Watch devices. In the game, players use the mobile device's GPS capability to locate, capture, battle and train virtual creatures, called Pokémon, who appear on the screen as if they were in the same real-

world location as the player. It quickly became a global phenomenon and was one of the most used and profitable mobile apps in 2016, having been downloaded more than 500 million times worldwide. It was credited with popularizing location-based and augmented reality gaming, promoting physical activity, along with helping local businesses grow. However, it attracted controversy for contributing to accidents and becoming a public nuisance at some locations. Various governments also expressed concerns over the security of the game, with some countries passing legislation to regulate its use.

Figure 1.3: In Pokémon Go, players must physically travel to explore the game's map and find Pokémons.

If, on the one hand, large IT companies are pushing more and more their LBSs without worrying too much about user privacy, on the other hand, researchers are investigating how to provide such services while preserving user privacy. In particular, various peer-to-peer overlay schemes that enable completely decentralized LBSs

have been presented, as described in next section. These peer-to-peer protocols, in addition to safeguard privacy of users inasmuch the data are not in the hands of a single possibly untrustworthy company, support the realization of bottom-up LBSs, not requiring large and expensive infrastructures.

1.2 Peer-to-Peer Architectures

Traditional peer-to-peer overlay schemes, such as Kademlia [9] or Chord [10], are not particularly suitable for location-based services, since they completely ignore geographic distances between peers of the network. The geographic distance, however, assumes an aspect of remarkable importance for location-based services, since they consume and produce geolocated information. For these reasons, the design of peer-to-peer overlay schemes proper to the development of location-based services is an extensively discussed issue in literature. Table 1.1 lists the main location-aware peer-to-peer overlays, sorted chronologically, and compares them based on the main features that location-based services should have. These peer-to-peer overlays are shown in next subsections.

1.2.1 GeoPeer

In GeoPeer by Araújo and Rodrigues [11], network peers arrange themselves to form a Delaunay triangulation, with the addition of long range contacts. GeoPeer is capable of providing some of the fundamental operations of location-based services, such as geographical multicast and queries. The main weaknesses of this work stem from the fact that only stationary peers are taken into account — which is a serious deficiency, since location-based services have to cope with devices that are mobile by definition.

1.2.2 Globase.KOM

A second notable solution is Globase.KOM by Kovačević, Liebau and Steinmetz [12]. This solution adopts a superpeer-based overlay, where each superpeer manages one of the rectangular zones that constitute the hierarchical layers in which the ge-

	Mobility of peers	Speed and direction of peers	Geographic broadcast	Completely peer-to-peer	Publish/Subscribe mechanism	Adaptive overlay scheme
GeoPeer			✓	✓		
Globase.KOM			✓			
DGT	✓			✓		
Geodemlia			✓	✓		
Overdrive	✓	✓	✓	✓		
ADGT	✓	✓	✓	✓	✓	✓

Table 1.1: Comparison of location-aware peer-to-peer overlay schemes.

ographic space is partitioned. Clearly, making use of a not completely distributed architecture implies a great effort in terms of complexity and a significantly poor scalability. Moreover, similarly to GeoPeer, this solution considers only stationary peers. Thus, it completely ignores the drastic increase of the maintenance overhead that would result by applying this protocol to mobile nodes.

1.2.3 DGT

Picone *et al.* describe a structured overlay scheme where each participant can efficiently retrieve peers or resources located near any chosen geographical location [13]. In such a system, called Distributed Geographic Table (DGT), the main pro-

vided service is to route requests to find available peers in a specific area. The DGT has been designed with primary emphasis on peers' mobility: each peer maintains a set of logic concentric circles, around its own geographical location, and keeps them constantly updated, in order to have the latest neighbors' location. Although the DGT takes into account the mobility of peers, it does not consider speed and direction of the motion. In effect, the mechanism by which each peer of this overlay scheme maintains updated its neighborhood completely ignores the different ways a peer can move.

1.2.4 Geodemlia

Geodemlia, by Gross *et al.*, is a peer-to-peer overlay that allows users to search for location-based information around specific geographic locations [14]. The Geodemlia overlay scheme is inspired by Kademlia [9] and provides geographical methods for search and store. Also, like the DGT, it uses concentric circles to divide the geographical space. However, this overlay completely overlooks peers that change their geographical location.

1.2.5 Overdrive

Heep *et al.* developed Overdrive, a peer-to-peer overlay very similar to the DGT, but considering also speed and direction of peers even if, actually, it uses information of speed and direction only to reduce the number of sent messages and not to enhance the overlay itself [15]. Differently from the DGT, Overdrive adopts a recursive approach for routing instead of the iterative one inspired by Kademlia that the DGT adopts.

Examining these works in literature, we have noticed that none of them really adapt the topology of the network based on peers' movements. We consider it an essential feature proper to a location-based service. For these reasons, we have extended the DGT peer-to-peer overlay in order to readjust itself considering the speed and direction of the peers. Moreover, we have adopted a recursive algorithm for message routing. Finally, we have provided a publish/subscribe mechanism by which

each peer can be automatically notified about any type of information around any geographical location, by simply exhibiting interest. All these features characterize our peer-to-peer overlay scheme for location-based services: the Adaptive Distributed Geographic Table (ADGT).

Chapter 2

Adaptive Distributed Geographic Table

With the objective to fulfill all the requirements of a location-aware peer-to-peer overlay scheme, we have redesigned and improved the DGT, thus obtaining the Adaptive Distributed Geographic Table (ADGT) [16]. In particular, to make the ADGT take fully into account peer mobility, we have formalized a new data structure for neighborhood management, different from all those available in literature. Such a new data structure is based on the idea that a peer should directly connect to those peers from which it is most likely to obtain satisfactory contents. In this way, an adaptive topology that reacts to peers' movements is obtained. Furthermore, we have switched from the traditional Kademlia-like iterative routing algorithm to a more efficient recursive one. To achieve this objective, we have changed the operations of discovery of other peers in the network. Finally, we have introduced the mechanism of subscription to any type of information around any geographical location, making the ADGT a complete peer-to-peer overlay scheme for LBSs.

2.1 General Principles

Let us denote the set of peers as $\mathscr{P}$, the space of the identifiers as $\mathscr{I}$ and the space of geographic coordinates as $\mathscr{W}$. Every peer is unambiguously described by a unique identifier $id \in \mathscr{I}$ and an ordered pair $w = \langle latitude, longitude \rangle \in \mathscr{W}$ representing a geographic location. More precisely, a generic peer $p \in \mathscr{P}$ is associated to the $\langle id_p, w_p \rangle$ pair, where $id_p \in \mathscr{I}$ and $w_p \in \mathscr{W}$.

To set the distance between any two peers, the *great-circle distance* is adopted. Such a metric is the shortest distance between two points on the surface of a sphere, measured along the surface of the sphere itself: $d : \mathscr{W} \times \mathscr{W} \rightarrow \mathscr{R}$. In the ADGT context, the sphere is the approximation of the Earth surface.

Given a geographical location, its *neighborhood* is defined as the set of peers that are geographically close to that location. More precisely, the neighborhood is the set of peers that are located inside a given region that surrounds the location of interest. Let us define $\mathscr{A}$ as the set of geographic regions delimited by a closed curve and $GB_w \in \mathscr{A}$ as a region centered in the geographic location w. Then, we define the neighborhood of w as $\mathscr{N}_w = \{p \in \mathscr{P} | w_p \subseteq GB_w\}$, where w_p is the geographical location of peer $p \in \mathscr{P}$.

2.2 GeoBucket Data Structures

Typically, in LBS-oriented peer-to-peer networks, neighborhood is maintained within circular regions whose center is each peer's geographical location. Thus, any direction equally matters. Such an approach is meaningful if we consider static or slowly moving peers. For example, a vehicle moving in the streets of the city center has to maintain its neighborhood within a circular region as interesting information may come from any direction. However, when a vehicle travels on the highway or any long road allowing for high speed, the interesting information is located along the travel direction. Thus, in such a scenario, a neighborhood bound within a circular region is useless. It is better to extend the region of interest forward and backward, with respect to the vehicle, rather than laterally.

Based on this principles, ADGT peers adapt their neighborhood regions by taking into account their own mobility, *i.e.*, their direction and speed. Given a location w, let us define a GeoBucket as a list of peers whose distance from w lies within a given interval. The GeoBucket is sorted by the distance from w. Every peer may maintain several concentric GeoBuckets, all of them being centered in the peer's location or in any location of interests. Nothing prevents one peer to maintain several sets of concentric GeoBuckets, each set having a different center. For example, one set of GeoBuckets may be centered in the peer's own location, while other GeoBuckets may be centered in locations of interest for the peer.

GeoBuckets are periodically updated, in order to maintain consistent information about neighbors' locations. As illustrated in Figure 2.1, the shape of the GeoBuckets boundaries is adaptively elliptical, rather than statically circular. More precisely, every GeoBucket has elliptical lower and upper bounds, where both semi-axes of the ellipses are computed according to the direction and speed of the target (which is the center of the ellipses). The set of GeoBuckets centered in a given location is defined by a group of K concentric ellipses. Every ellipse has a semi-major axis a_i and a semi-minor axis b_i, with i integer $\in [1,K]$:

$$a_i = i \cdot t_i \cdot (1 + S \cdot \frac{v}{V_{max}})$$

$$b_i = i \cdot t_i \cdot \frac{V_{max}}{V_{max} + S \cdot v}$$

where v is the current speed of the target, V_{max} is the maximum allowed speed, t_i is the thickness of each GeoBucket and S relates the shape of the GeoBucket to the current speed of the target. In particular, the higher the speed, the higher the eccentricity of the ellipses. Moreover, the direction of the semi-major axis coincides with the moving direction of the target. Conversely, when the target is stationary ($v = 0$), the ellipses reduce to circles.

Using the formal notation introduced in the previous subsection, the region covered by the GeoBucket structure centered in w is

$$GB_w = \{w' \in \mathscr{W} | d(w', f_1) + d(w', f_2) \leq 2 \cdot a\}$$

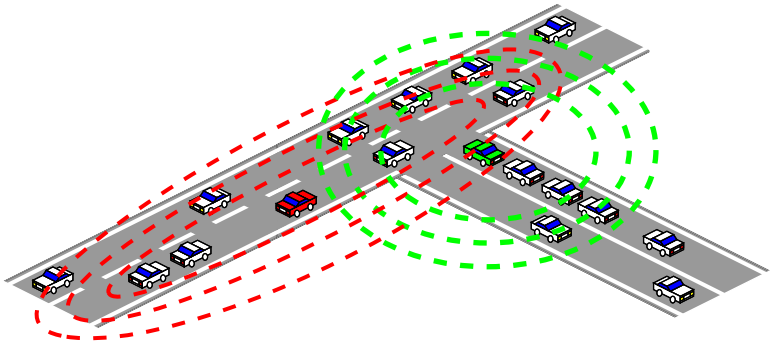


Figure 2.1: The ADGT adaptive elliptical neighbourhood.

where f_1 and f_2 are the two foci and a is the semi-major axis of the external ellipse centered in w.

2.3 Routing Strategies

The original DGT implementation used an iterative algorithm, very similar to the one adopted by Kademlia, during the discovery process of new peers. To improve the efficiency of the routing procedure [17], we have moved to a recursive implementation.

In particular, we have defined a discovery message characterized by a geographic location around which to search for new neighbors, as reported in Listing 2.1. When a peer wants to discover new neighbors, it chooses the closest peer among those it knows and sends it a discovery message specifying the geographical location of interest. If a peer receives a discovery message, it replies to the requester with a set containing the β closest peers to the specified geographical location, and forwards the discovery request to the closest peer to the specified geographical location it knows.

If the peer that receives the discovery message is the closest peer to the specified geographical location among its neighbors, the recursive discovery stops and the request is not forwarded.

```
function on(discoveryRequest) {

    var position = discoveryRequest.position;
    var sender = discoveryRequest.sender;

    // Retrieve at most "beta" peers from the GeoBucket, different from the sender of the discovery request
    var neighbors = geobucket.find(position, beta, sender);

    if(neighbors.length > 0) {
        var discoveryResponse = new DiscoveryResponse();
        discoveryResponse.set(neighbors);

        sender.send(discoveryResponse);

        var nearest = neighbors[0];

        // If there is a closer neighbor, the discovery request is forwarded
        if(distance(this.position, position) >= distance(nearest.position, position)) {
            nearest.send(discoveryRequest);
        }
    }
};
```

Listing 2.1: Recursive algorithm for peers discovery.

2.4 Publish/Subscribe Mechanism

One of the most important features that a location-based service must provide is the possibility to notify the user about something of his/her interest near any geographical location. For example, the service must be able to notify the presence of friends nearby the user, or the presence of traffic jams along the route that the user is traveling. Nevertheless, all previous works have made the assumption that a peer has to know only its geographical neighborhood, *i.e.*, those peers whose geographical location is not too far from that of the peer itself. This means that the peer can be notified about any type of information only if it is located in its vicinity. Therefore, a user could not be able to monitor traffic conditions along a particular city thoroughfare, without necessarily being close to it.

In the ADGT, we have introduced a publish/subscribe mechanism through which peers can be notified about any information they are interested in around any geo-

graphic location [18]. Our significant improvement is apparently simple: while in the original DGT every peer had a unique GeoBucket structure, now it can have multiple GeoBucket structures, associated with different locations. In this way, it is possible, for example, to place a GeoBucket structure on a particular road junction to be automatically notified about any related warning.

Obviously, this brings out changes in the maintenance strategies of the neighborhood. Previously, it was possible to state that if peer B belongs to the neighborhood of peer A, then peer A must be in the neighborhood of B, that is, if a peer A adds B to its GeoBucket structure, then B must also do the same. However, with the introduction of the possibility of being able to keep more GeoBucket structures for each peer, such an assumption is no longer possible. It is thus required a mechanism to allow a peer to inform another peer about the interest in receiving location updates.

To do this, every peer maintains a limited set of references to the last peers which have expressed interest in receiving location updates. Periodically, each peer iterates over its set and transmits location update messages to interested peers. If a peer receives a location update message from another peer about which it is no longer interested in, all it has to do is to inform the sender, that will remove it from its set.

Having multiple GeoBucket structures allows the peers to become aware of other peers near any geographical location, and so to query those peers about anything or to inform them about particular information they want to know, following the publish/subscribe model.

Actually, multiple GeoBucket structures make geographical broadcast possible. All that is needed is a GeoBucket structure that covers the area interested by the broadcast and disseminates messages to all the peers contained in the GeoBucket structure.

Chapter 3

Adgt.js Web Framework

As described in previous chapter, in recent years researchers have investigated to provide location-based services in a privacy-aware manner, presenting various peer-to-peer (P2P) overlay schemes that enable completely decentralized LBSs. Despite the many benefits of a P2P approach, often these solutions have been studied only in simulative environment and truly usable implementations have never been released.

In this chapter is presented a working implementation of the ADGT overlay scheme. The objective behind the development is the software interoperability between all possible and heterogeneous devices, to make sure that the adoption is high. For this reason, we turned to real cross-platform technologies, such as WebRTC, WebSocket and JavaScript to build a framework that supports the development of P2P-based LBSs. To the best of our knowledge, this implementation is the first of its kind in the area of P2P protocols for LBSs.

3.1 Open Web Platform Technologies

As the main idea behind our implementation is the complete interoperability among devices as much as possible different – both from the hardware point of view, and in terms of installed software – we have turned to those technologies that constitute

the Open Web Platform (OWP)[1]. The OWP is a collection of open royalty-free Web technologies, such as HTML5 and JavaScript, developed by the World Wide Web Consortium (W3C) and other Web standardization bodies such as the Unicode Consortium, the Internet Engineering Task Force (IETF), and ECMA International, with the objective to obtain a platform that works on all browsers, operating systems and devices, without requiring any approvals or waiving license fees.

Although the OWP standards have different maturity levels, and the development of most standards is still in progress, the web browser has become the main access interface to the Internet and has actually become synonymous with the Internet itself for a large portion of Internet users. While initially web browsers were designed only to display information provided by web servers, thanks to this standardization process, they are becoming the real cross-platform technology, being able to truly realize the "write once, run everywhere" unfulfilled promise of Java related technologies.

Among the many technologies that are encompassed under the umbrella of Open Web Platform, one of the most interesting definitions the W3C has worked on is the Web Real-Time Communication (WebRTC)[2], a free and open API that supports browser-to-browser applications for voice calling, video chat, and peer-to-peer data sharing without the need of either internal or external plugins. Its aim is to enable rich, high quality, real-time applications to be developed for browsers, mobile platforms, and IoT devices, allowing them to communicate via a common set of protocols. WebRTC, WebSocket API[3], Geolocation API[4] and ECMAScript[5] are the OWP technologies we have embraced to implement the ADGT protocols.

3.1.1 WebRTC

Nowadays, the Internet is no more a stranger to audio and video communication. Talking to someone over a voice/video call has become a simple task for an everyday

[1] *Open Web Platform* `https://www.w3.org/standards/`
[2] *WebRTC* `http://www.w3.org/TR/webrtc/`
[3] *WebSocket API* `http://www.w3.org/TR/websockets/`
[4] *Geolocation API* `http://www.w3.org/TR/geolocation-API/`
[5] *ECMAScript* `http://www.ecmascript.org/`

user, thanks to common programs such as Apple FaceTime, Google Hangouts and Skype. Together with these applications, a wide range of techniques and solutions to problems have been developed and engineered, such as packet loss, recovering from disconnections, and reacting to changes in network, to ensure high quality communications.

The purpose of WebRTC is to bring all of this technology into the browser. Differently from those solutions that require the installation of plugins which can be difficult to deploy, test and maintain, and may necessitate licensing fees from developers, WebRTC brings high-quality audio and video to the open Web.

Moreover, WebRTC supports data transfer: since a high-quality data connection is needed between two clients for audio and video, it also makes sense to use this connection to transfer arbitrary data. Indeed, WebRTC enables data streaming between browser clients without the need to install plugins or third-party software, implying a strong integration between the content presented by the browser and the real-time content. With WebRTC, web browsers become peers of a real peer-to-peer network, being capable to exchange data in an unmediated fashion.

To acquire and communicate streaming data, WebRTC implements the following APIs:

- `MediaStream`, which represents synchronized streams of media such as user's camera and microphone;
- `RTCPeerConnection`, which handles stable and efficient communication of streaming data between peers, with facilities for encryption and bandwidth management;
- `RTCDataChannel`, which enables peer-to-peer exchange of arbitrary data, with low latency and high throughput.

The `MediaStream` interface represents a stream of data of audio and/or video. A `MediaStream` may be extended to represent a stream that either comes from or is sent to a remote node, and not just the local camera. This API will not be detailed further here because it is not strictly relevant to the presented work.

The `RTCPeerConnection` interface models a WebRTC connection between the local computer and a remote peer. It is used to handle efficient data streaming between the two peers.

Unlike most web applications that choose the Transmission Control Protocol (TCP), WebRTC relies on User Datagram Protocol (UDP) as the default transport protocol. In fact, if on the one hand TCP guarantees delivery of data in the exact order and without duplication, on the other hand in streaming applications most data quickly become obsolete and, if any data were to be ensured in the reception, this would result into a bottleneck in case of data loss. Since a completely reliable connection is not a requirement for audio, video and data streaming transmissions, while a very fast connection between the two browsers is highly desirable, UDP has been chosen as the default transport protocol in WebRTC. In particular, WebRTC transports audio and video streams using the Secure Real-Time Transport (SRTP) protocol, which is real-time, and provides encryption, message authentication and integrity to transmitted data. `RTCPeerConnection` hides all the complexities of WebRTC to web developers. WebRTC uses codecs and protocols to make real-time communication possible, even over unreliable networks, adopting techniques for packet loss concealment and noise reduction and suppression, in a completely transparent manner to developers.

Another feature that `RTCPeerConnection` offers to web developers is the Interactive Connectivity Establishment (ICE), a technique developed by the Internet Engineering Task Force [19] to overcome the complexities of real-world networking, where most devices live behind one or more NAT layers, some have anti-virus software that blocks certain ports and protocols, and many are behind proxies and corporate firewalls. First, ICE tries to make a connection using the host address obtained from the operating system and the network card. In case of failure, ICE uses the Session Traversal Utilities for NAT (STUN) [20] protocol to discover the public address of the device and then pass that on. If also this attempt fails and a direct communication between peers over UDP cannot be established, ICE falls back on Traversal Using Relays around NAT (TURN) [21], rerouting the traffic via a TURN relay server using TCP.

	TCP	UDP	SCTP
Reliability	reliable	unreliable	configurable
Delivery	ordered	unordered	configurable
Transmission	byte-oriented	message-oriented	message-oriented
Flow control	yes	no	yes
Congestion control	yes	no	yes

Table 3.1: SCTP is a transport protocol, similar to TCP and UDP, which can run directly on top of the IP protocol. However, in the case of WebRTC, SCTP is tunneled over a secure DTLS tunnel, which itself runs on top of UDP [1].

The `RTCDataChannel` interface allows us to transfer arbitrary data directly from one peer to another. It works with the `RTCPeerConnection` API, which enables peer-to-peer connectivity with lower latency, and uses Stream Control Transmission Protocol (SCTP), allowing configurable delivery semantics: out-of-order delivery and retransmit configuration.

SCTP is a transport-layer protocol, serving in a similar role to the popular protocols TCP and UDP that provides some of the same service features of both: it is message-oriented like UDP and ensures reliable, in-sequence transport of messages with congestion control like TCP.

`RTCDataChannel` can work in either reliable mode (analogous to TCP) or unreliable mode (analogous to UDP). The first guarantees the transmission of messages and also the order in which they are delivered. This takes extra overhead, thus potentially making this mode slower. The latter does not guarantee every message will get to the other side nor what order they get there. This removes the overhead, allowing this mode to work much faster.

Furthermore, in the case of WebRTC, SCTP sits on top of the Datagram Transport Layer Security (DTLS) protocol, which is derivative of SSL, and provides communication security for datagram protocols. In particular, using DTLS, WebRTC guarantees that every peer connection is automatically encrypted and, in particular:

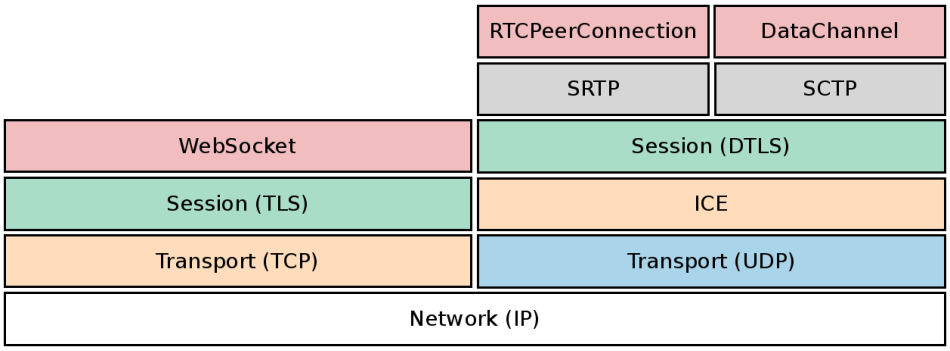


Figure 3.1: WebRTC and WebSocket protocol stack.

- messages are not readable if they are stolen while in transit between peers;
- a third party cannot publish messages within the ADGT overlay network;
- messages can not be altered while in transit;
- the encryption algorithm is fast enough to support the highest possible bandwidth between peers.

3.1.2 WebSocket

The Web has been traditionally tied to the request/response paradigm of HTTP. Nevertheless, with the need to have a more and more dynamic web, technologies such as AJAX have emerged. However, all of these technologies are not well suited for low latency applications, because of the HTTP overhead.

The WebSocket specification defines an API establishing an interactive communication session between a web browser and a server. With this API, the client and the server can make a persistent full-duplex connection between them and send data to each other at any time. The main advantage is that the client can send messages to a server and receive event-driven responses without having to poll the server for a reply.

3.1.3 Geolocation

The Geolocation API defines a high-level interface to location information associated with the device. The API itself is agnostic of the underlying location information sources: location can be indiscriminately obtained from a Global Positioning System (GPS), inferred from network signals such as IP address, RFID, Wi-Fi and Bluetooth MAC addresses, and GSM/CDMA cell IDs, as well as user input.

The API provides the location information represented by latitude and longitude coordinates. The API is designed to enable both "one-shot" position requests and repeated position updates, as well as the ability to explicitly query the cached positions.

3.1.4 ECMAScript

ECMAScript is a scripting language specification standardized by ECMA International. JavaScript is one of the most known implementation of the language.

The current version of the ECMAScript Language Specification standard is ECMAScript 2015 (6th Edition) and introduces language support for classes, constructors, and the `extend` keyword for inheritance. Moreover, it provides a way to load and manage module dependencies, new `Map` and `Set` objects, `Promise` objects and many other features.

3.2 Implementation

We have implemented the ADGT protocol using OWP technologies only [22]. The resulting ECMAScript 6 framework, denoted as Adgt.js, can be freely used for the realization of peer-to-peer LBSs, where it is important to discovery geographic neighbors and exchange messages with them using a technology that guarantees security and data encryption. Being a framework, Adgt.js gives a way to flexibly organize the code of location-based applications, and it is not simply a set of useful predefined functions.

In particular, we have defined a JavaScript `Peer` class that represents the ADGT peer. This class is characterized by a `Descriptor`, *i.e.*, a composition of an unique

identifier of the peer in the network and its geographic location. This latter implements the `Position` interface defined in the Geolocation API and represents the position of the peer at a given time, but also its altitude and its speed.

Furthermore, the `Peer` class contains a reference to a `GeoBucket` object that, as the name says, implements the peculiar routing table of the ADGT protocol. Our GeoBucket implementation consists of a wrapper of the new ECMAScript 6 `Set` class, whose elements are nodes of the network. The `GeoBucket` class, in addition to being a collection of nodes, presents functionalities for the management of geographic neighborhood, therefore to add and remove nodes that approach and move away from the peer, and to update the information about the geographic locations of the neighbors.

In Adgt.js, neighbors are represented by the `RemoteNode` class, which actually realizes the peer-to-peer connection with other peers, through WebRTC technologies. More specifically, this class allows to connect to another peer of the ADGT overlay network using the `RTCPeerConnection` interface, and to directly send a message to it with the `DataChannel` interface. In this way, all data exchanges between network nodes — such as position updates as well as peer discovery messages — are realized using WebRTC.

`DataChannels` are also used as signaling channels. In fact, signaling methods and protocols, *i.e.*, the mechanisms required to coordinate communication and to send control messages, are not specified by WebRTC. WebRTC assumes the existence of a communication coordination process, allowing clients to exchange session control messages (outlined by the JavaScript Session Establishment Protocol [23]), error messages, media metadata such as codecs and codec settings, bandwidth and media types, key data, used to establish secure connections, and network data, such as a IP address and port, without placing constraints on the signaling technology.

Although a signaling service consumes relatively little bandwidth and CPU per client, signaling servers for a popular application may have to handle a lot of messages, from different locations, with high levels of concurrency. For this reason, we have decided to decentralize the responsibility to act as signaling servers among all the peers of the network, using `DataChannels`. In particular, the peer discovery

operation has been realized in a way that when a peer receives the list of neighbors from the peer that has contacted, the latter acts also as a signaling server between the first and the possible peers which have to be contacted.

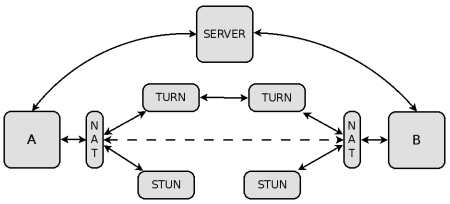


Figure 3.2: Using ICE to cope with NATs and firewalls.

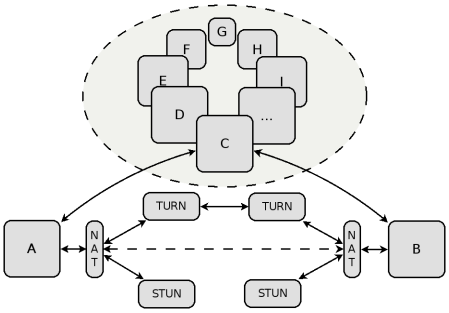


Figure 3.3: Peers of the network act as signaling servers in our implementation.

A direct connection between two peers can be achieved by means of a signaling server that coordinates the communication. Actually, a signaling server by itself is not sufficient to overcome the complexities of real-world networking, whereas it can be solved with the use of ICE technology, as shown in Figure 3.2.

To increase system scalability, Adgt.js has been designed to be architecturally different from what depicted in Figure 3.2, since the operations of signaling between two peers that are attempting to establish a connection are provided by an intermediary peer rather than from a centralized server. Figure 3.3 represents the architecture of our implementation.

In particular, the peer designed to act as a signaler between two other peers is the one that allowed the other two to get to know each other, at the end of the discovery process. Figure 3.4 shows the sequence of messages exchanged during a discovery operation, in the event that peer *A* wants to start a conversation with peer *B*, just discovered by means of peer *C*. After peers *A* and *C* have exchanged discovery messages, where *A* asks for a specific geographic location and *C* returns a list of known peers near the location indicated including *B*, if peer *A* wants to add peer

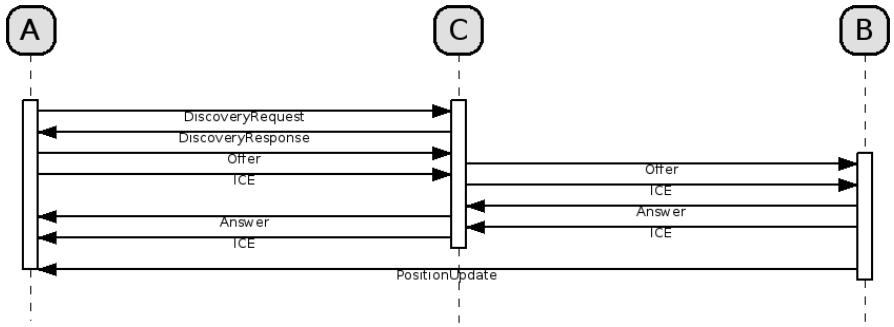


Figure 3.4: Messages exchanged during discovery operation.

B to its GeoBucket, peer *C* will be the signaler among them. The first message that *A* has to send to *B* through *C* is an `Offer` message, which is a serialized session description message, followed by an `ICE` message with the information about network interfaces and ports. On the other side, when *B* receives the `Offer` message, it replies to *A* through *C* with an `Answer` message containing its session description, therefore with an `ICE` message. Finally, at the end of this initialization, *A* and *B* can directly exchange ADGT or application-specific messages, such as updates on their geographic location. Despite the complexity of the architecture, the implementation hides all these aspects to users, which do not have to worry how the connections are established.

Since it is not always possible to use another peer as a signaling server, *e.g.*, when the peer joins the peer-to-peer network, each peer has a reference to a `BootstrappingNode` that is able to operate as a signaling server for those peers that log on to the network for the first time (Bootstrapping and other peer-to-peer patterns are illustrated in a recent work by Amoretti and Zanichelli [24]). We have built this type of signaling server using the Node.js framework[6] and the WebSocket protocol. The choice of adopting Node.js, which is an open source runtime environment based on Google's V8 JavaScript engine, has allowed to reuse most of the code written for the ADGT implementation. Furthermore, the WebSocket protocol allowed us to

[6] *Node.js* `https://nodejs.org`

encrypt the signaling and negotiation communication like the standard HTTPS protocol works, ensuring that no one can intercept messages sent to the server to figure out which peers are talking to whom.

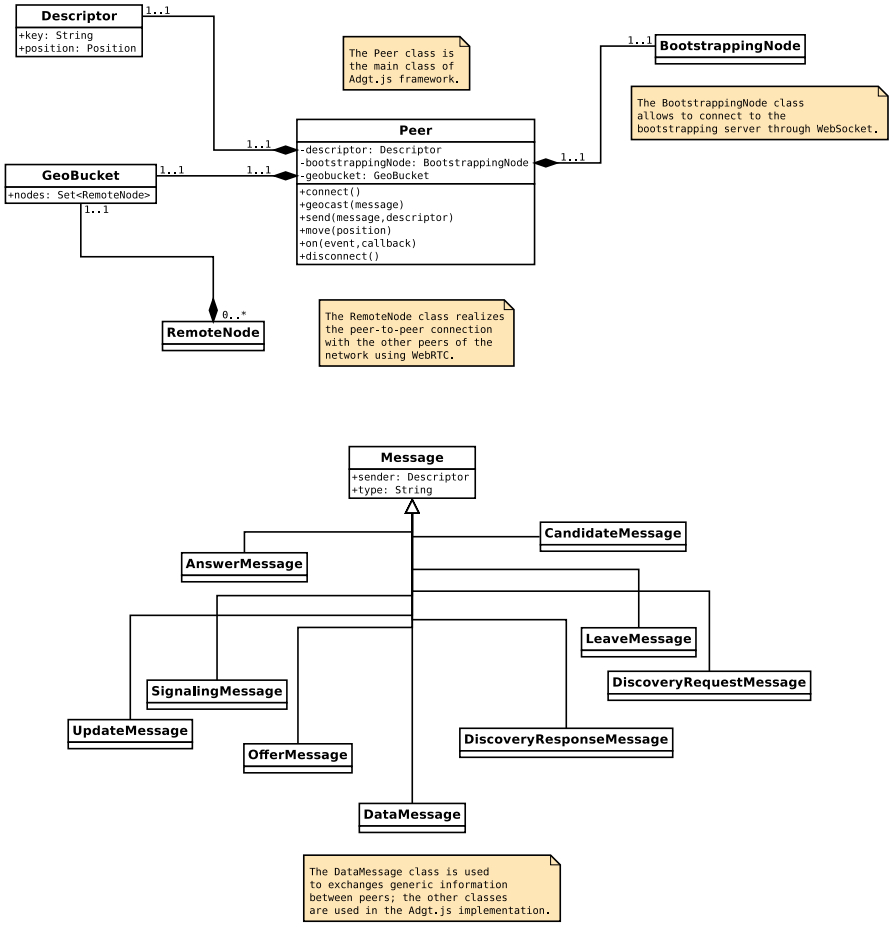


Figure 3.5: Class Diagram of the Adgt.js framework.

Figure 3.5 describes the structure of the system by showing the classes, their attributes, operations (or methods), and the relationships among objects of the Adgt.js framework.

Adgt.js has been released online[7] with a free and open source software license and can be used as a web application framework without restrictions.

3.2.1 Algorithm for GeoBucket Collision Detection

In order to identify two geographically close peers, *i.e.*, neighbors in the peer-to-peer overlay network, we have devised an efficient algorithm for verifying the collision of elliptical GeoBuckets. In particular, the algorithm checks if two ellipses represented in the geographic coordinate system overlap.

To obtain this, we have developed a simple method that approximates each ellipse in a polygon inscribed in the ellipse itself. In particular, this method starts from the rhombus that connects the four vertices of the ellipse and iteratively calculates geographical coordinates of points that are on the perimeter of the ellipse itself. Once these points, whose number depends on the number of iterations (*i.e.*, on the level of accuracy one wants to achieve), have been determined, the method draws the polygon that connects them and that actually is projected on the spherical surface of Earth.

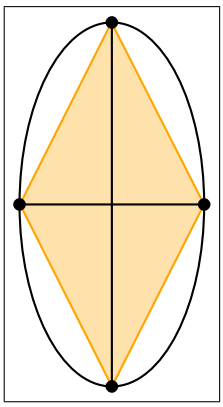


Figure 3.6: Step 1: initial polygon (rhombus) embedded in the ellipse.

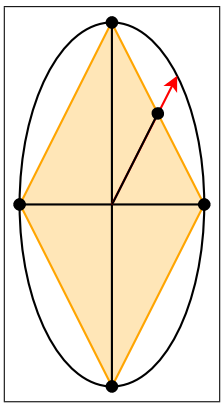


Figure 3.7: Step 2: calculation of the vertex of the new polygon.

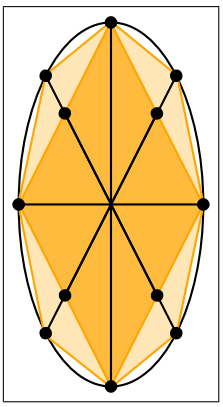


Figure 3.8: Step 3: the new polygon embedded in the ellipse.

[7] *Adgt.js* `https://github.com/brambilla/adgt.js`

Figures 3.6, 3.7 and 3.8 show the sequence of operations performed by the algorithm in order to obtain a polygon embedded in the ellipse of GeoBucket.

The algorithm proceeds by checking if there is any overlap between the two polygons that approximate the ellipses of the GeoBuckets.

Chapter 4

Evaluation through Simulation

It is an established fact that in the immediate future there will be billions of objects with the capability to communicate and sense or interact with their internal states or the external environment [25]. These pervasive systems and applications, due to their complexity, need careful analysis and test, before being deployed to target environments. Consider, for example, smart city applications [26], requiring the coordination of a huge number of networked software entities, interfaced with sensors, actuators, computational and storage facilities.

In this chapter, we describe our cost-effective software-in-the-loop simulation methodology and present the Java-based simulation platform we have realized, which allows to simulate interconnected devices easily by providing an intuitive simulation methodology, which results in maximal code reuse. The proposed platform provides a general-purpose simulation engine, which includes specific packages to simulate mobility, networking, and energy consumption models. Furthermore, the proposed methodology allows to define general-purpose devices, which can be characterized by multiple network interfaces and protocols, as well as different network and energy models. With this solution, we can easily tackle problems like flexibility, modularity, code reuse and ease of deployment.

Thanks to the simulation environment we have realized, it has been possible to deeply evaluate the ADGT in different scenarios of mobility, obtaining very promis-

ing results presented in Section 4.3 of this chapter.

4.1 Related Work

The Internet of Things (IoT) mainly refers to the interconnection of a multitude of constrained devices with limited computational and memory capacity, typically battery-powered. As a consequence, energy efficient technologies must be adopted. Moreover, these devices usually operate in constrained networks, which often have high packet error rates and a throughput of tens of kbit/s. IoT applications require a huge number of networked devices, equipped with sensors, actuators, computational and storage facilities, to cooperate and be coordinated. Due to their complexity, such applications need careful analysis and testing, before being deployed to target environments. However, testing them in a controlled environment, such as a laboratory, may not be sufficient to understand and evaluate the possible properties/issues of such complex systems and, in particular, *emergent* ones, *i.e.*, those that cannot be inferred from the analysis of single components, and appear when components interact. Moreover, the simulation of large-scale systems is usually based on approximated models and simulation-specific code, which are not representative of all system details.

In this context, standardization organisms, such as the Internet Engineering Task Force (IETF) and IPSO Alliance, have been working on standard and interoperable communication mechanisms, in order to interconnect these devices. In particular, the Internet Protocol version 6 (IPv6) [27] has been identified as the main candidate, and several working groups have been set in order to address the issues related to IoT devices communication. For example, the IETF IPv6 over Low power WPAN (6LoWPAN) Working Group [28] is defining encapsulation and other adaptation mechanisms in order to send and receive IPv6 packets over Low power Wireless Personal Area Networks, such as those based on IEEE 802.15.4. From the point of view of the application layer, the IETF Constrained RESTful Environments (CoRE) Working Group[1] is currently defining a Constrained Application Protocol (CoAP) [29], a software protocol designed to easily translate to HTTP for simplified integration with

[1] *Constrained RESTful Environments Working Group* `https://tools.ietf.org/wg/core/`

the web, while also meeting specialized requirements such as multicast support, very low overhead, and simplicity, intended to be used in resource-constrained Internet devices, such as wireless sensor network nodes.

Accurate simulation of sensors taking part in wireless sensor networks is often coupled with the operating system running on top of the sensor. Most of the specialized IoT operating systems typically provide simulation environments for developers: for instance, Cooja [30] and TOSSIM [31] are the simulation platforms for testing applications running on Contiki OS [32] and TinyOS[2], respectively. The approaches of both simulators totally differ from generic simulation frameworks like OMNeT++ [33] or ns-3[3], since they simulate the entire behavior of the node, from hardware and communication to the whole software stack running on the operating system, which precludes the possibility to simulate a high number of deployed nodes.

These platforms are targeted at evaluating specific aspects (*e.g.*, link- or application-layer protocols and energy consumption), other frameworks, such as ns-3, provide a much wider variety of network and communication protocols and Internet system representations. However, the entire domain of wireless sensor networks has been affected by the increasing importance of the Internet of Things and related technologies, such as 6LowPAN [28]. As standardization efforts are being carried out, resulting in the design of standard communication protocols on top of IP, developers are starting to out the ever-increasing need for simulation platforms, which can be used to test large-scale systems comprising a high number of nodes, without taking into account low-level issues (*i.e.*, hardware platforms and operating systems), but focusing on application-specific issues. Weingärtner *et al.* in [34] report the need for an accurate IoT simulator and discuss different approaches to overcome the limitations of available simulators. Among all the approaches, a hybrid simulation environment, based on a combination of available generic frameworks and system-level simulators together, might reduce the gap between research and practical deployments and strengthen the option of simulative studies for the IoT area.

In [35], the author highlights the need for interconnection between the two differ-

[2] *TinyOS* `http://www.tinyos.net`

[3] *ns-3 - Discrete Event Network Simulator* `http://www.nsnam.org`

ent types of network simulators and suggests the integration of an accurate operating system simulator into a generic simulation environment with a correct representation of protocols from the different layers of the used network stacks. The MAMMoTH project aims at building an emulation platform that can support nodes in the order of tens of millions [36], but it has not been recently updated.

In this context, various programming frameworks have been proposed to support the development of pervasive computing applications [37, 38, 39]. However, they require fully-equipped pervasive computing environments.

The development of software modules to be installed on resource-constrained devices, such as nodes of wireless sensor networks, is often supported by device emulators [32]. When the testing phase is completed, the application code can be installed on real devices as is, and its logic does not require further debugging. Unfortunately, the number of nodes that can be emulated at the same time is usually very limited. Thus, testing in the large requires simulation.

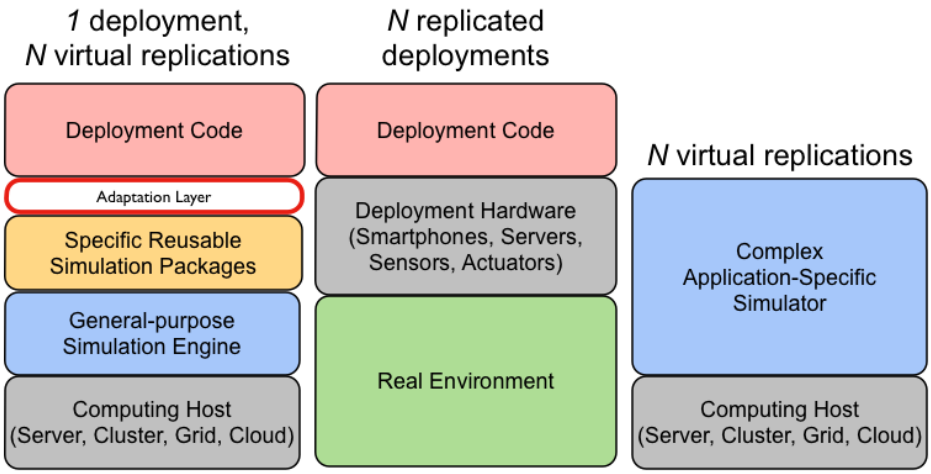


Figure 4.1: Layered representation of the proposed software-in-the-loop simulation methodology, compared with traditional simulation and on-field testing.

Bruneau and Consel have recently proposed the DiaSim simulator [40], which targets applications based on sensors and actuators, deployed in physical environ-

ments, involving users. DiaSim enables the simulation of the application logic of such applications, but does not simulate the other components of a pervasive computing environment. *E.g.*, it does not provide any support to estimate physical aspects of an environment (such as the thermal modeling of a room), to simulate the network traffic between application components, or to model the behavior of application users. DiaSim is parameterized with respect to a high-level description of the target pervasive computing environment.

PerSim is an event-driven, interactive simulator, which allows to design a pervasive space and fit it with the desired sensors relevant to a specific scenario or application [41]. The simulator provides a web-based interface to enable incremental (long-live sessions) and collaborative design of pervasive space simulation projects. Researchers can design the space in terms of sensors, actuators, activities, and effectively generate data by changing and fine tuning simulation parameters. The authors have proposed an XML-based standard for Sensory Dataset Description Language (SDDL), alternative to the more common Resource Description Framework (RDF).

With respect to these solutions, our approach focuses on

1. *seamless integration of deployment code* over simulated devices and environments;

2. *high modularity*, as specific simulation packages can be plugged/unplugged over a general-purpose simulation engine;

3. *scalability*, as the number of replicated devices can be deterministically or stochastically defined and increased/decreased by the simulation engine, according to high-level system specifications.

4.2 Proposed Methodology

The software-in-the-loop simulation methodology we propose allows to test deployment software on simulated devices, immersed in simulated environments. On top of the computing host, a general-purpose simulation engine is installed. More specific

simulation packages are installed, and integrated with deployment software for pervasive applications. The only effort that is required, for each different test scenario, is to develop lightweight adapters to integrate deployment code with simulation packages.

Figure 4.1 compares the proposed software-in-the-loop simulation methodology (illustrated by the layered diagram on the left) with on-field testing (shown by the diagram in middle) and traditional simulation (presented by the diagram on the right). The main advantage of software-in-the-loop simulation is that it requires just one deployment of the software to be tested, as the simulation engine generates and manages *N* virtual nodes, providing the illusion of multiple replications [42]. On-field testing, conversely, requires *N* deployments of the code to be tested. Traditional simulation, instead, provides *N* virtual nodes with virtual code, *i.e.*, a simplified version of the deployment code. Usually, traditional simulation is convenient in terms of execution time, with respect to software-in-the-loop. However, its reliability is much lower.

For example, consider a point-to-point message transmission, requiring the execution of a `send(msg)` function provided by a specific socket-based API, such as Sip2Peer[4]. In this case, the adapter should expose the same function signature, but the related implementation would not be a socket-based message transmission. Instead, it should be an event scheduling on the simulation engine, with a timestamp in the future, computed according to a realistic delay model.

The proposed methodology is based on the concept of *IoT Node*, which represents a generic smart object endowed with a mobility model, one or more network models, and an energy model, which can interact and provide feedback among themselves to better characterize the behavior of simulated nodes. The mobility model defines the movement of the IoT Node, and how its location, velocity, and acceleration change over time. Network models describe network capabilities of each interface the IoT Node is equipped with. For example, these models define delays and failure rates in the delivery of data in the network. Finally, the energy model describes the behavior of the IoT Node from an energy consumption point of view and can also take into account duty-cycling.

A visual model of the IoT Node is shown in Figure 4.2. Network models are

[4] *Sip2Peer* `https://github.com/dsg-unipr/sip2peer`

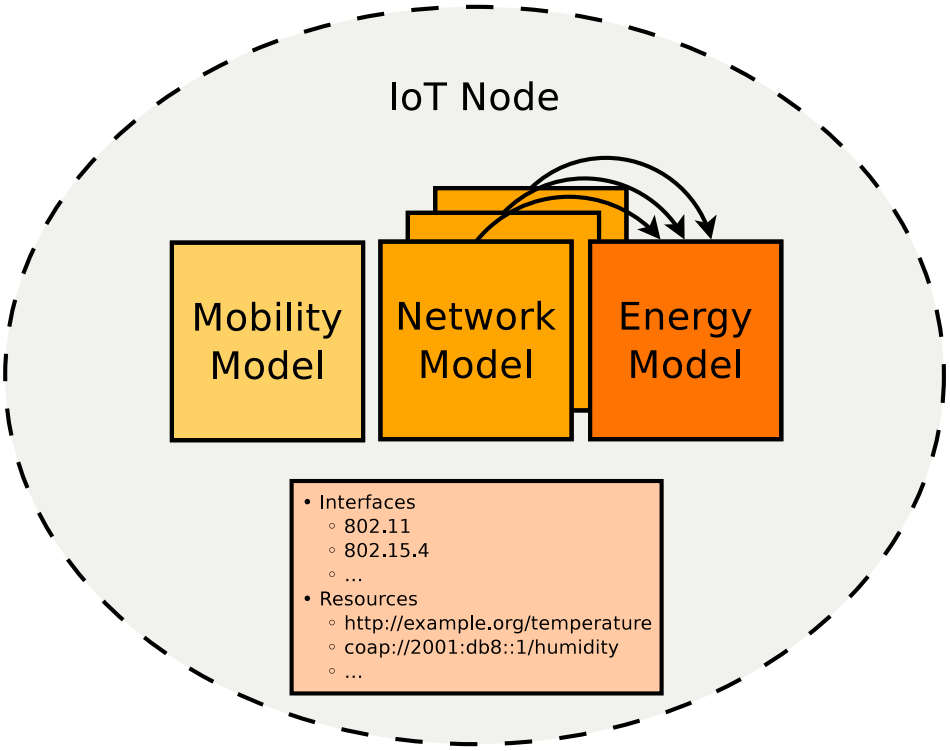


Figure 4.2: Visual model of an IoT Node.

linked to the energy model, since the use of network interfaces affects energy consumption, which varies depending on the type of access network technology.

An IoT Node is also characterized by

1. different network interfaces (*e.g.*, IEEE 802.15.4, IEEE 802.11, IEEE 802.3, or Bluetooth);

2. various communication application-layer protocols (either standard, such as CoAP [29], HTTP [43] or MQTT-SN [44], or user-defined, such as CoSIP [45]);

3. resources, which may be identified by their Uniform Resource Identifier (URI).

Through such a definition of IoT Node, our simulation platform reveals a high flexibility: it is possible to simulate different applications which make use of IoT devices characterized by any mobility, network and energy models. Furthermore, being composed of models with a loose coupling, our IoT Node has a high modularity and allows the reuse of most of the code. The replacement of the energy model does not require to modify the others, for example. Moreover, by adopting the required interfaces, it is possible to directly deploy the source code on real devices.

4.2.1 Architecture

Starting from the concept of IoT Node, the main class of our simulation platform: the `IoTNode` class has been defined. By extending this class, developers can implement and simulate IoT applications. The extension of generic IoT Nodes allows to isolate the application layer from the underlying layers and brings the advantage to focus on application-specific aspects and, reuse the entire application logic (and implementation, where applicable).

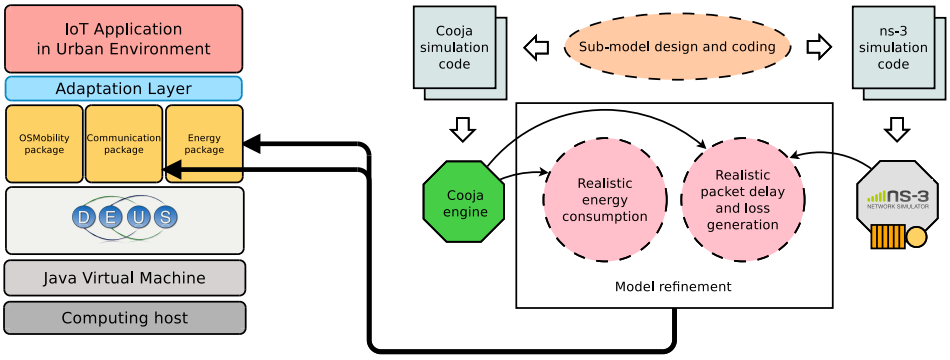


Figure 4.3: Layered representation of the proposed methodology.

The architecture of the proposed simulation platform is shown in Figure 4.3. The architecture is modular and based on several technologies. The top layer of the architecture, depicted in Figure 4.3, represents the application to be tested, whereas the underlying layer (denoted as *Adaptation Layer*) is intended as the coordination of the `IoTNodes`. The remainder of the architecture and the main features of core

components of our simulation framework are detailed next.

4.2.2 DEUS

Since the objective is the simulation of a high number of IoT Nodes, we searched for a generic simulation engine and the choice fell on DEUS [46] for its high scalability and versatility. DEUS is a general-purpose discrete event simulation environment. It is a free software project developed in Java. Moreover, DEUS supports parallel (multicore and/or distributed) simulations [47]. Its APIs allow developers to implement (by sub-classing) (i) *nodes*, *i.e.*, the entities which interact in a complex systems, leading to emergent behaviors; (ii) *events*, *e.g.*, node births and deaths, interactions among nodes, interactions with the environment, logs and so on; and (iii) *processes*, either stochastic or deterministic ones, constraining the timeliness of events.

Furthermore, it exists an extension of DEUS dedicated to the modeling of mobile nodes that maintains its characteristics of generality and abstraction. Such an extension, called OSMobility, allows to easily realize and model the aspects related to mobility of IoT Nodes [48].

4.2.3 OSMobility Package

OSMobility[5] is a simulation environment which allows to simulate the motion of different entities, such as pedestrians and vehicles, in realistic geographical spaces. Being based on DEUS, OSMobility inherits all the features that make it versatile and generic.

The main class of OSMobility is `GeoNode`, which is a generic element of a simulation, and is characterized by a geographical location.

It is extended by `StationaryNode` and `MobileNode`, which represent a static node (*e.g.*, a traffic light, roadside sensor) and a mobile node, respectively. OSMobility is integrated with OpenStreetMap (OSM)[6], an open database which provides geographical data, such as road maps, for free, and uses such data to compute

[5] *OSMobility* `https://github.com/brambilla/osmobility`
[6] *OpenStreetMap* `https://www.openstreetmap.org/`

node trajectories, with a resolution degree which ranges from millimetres to kilometres, also taking into account speed limits and points of interest. Thus, OSMobility allows to simulate vehicles running on highways or urban roads, pedestrian walking within Limited Traffic Zones, bicycles moving on cycling lanes, etc.

Thus, OSMobility allows to simulate vehicles running on highways or urban roads, pedestrians walking within Limited Traffic Zones, bicycles moving on cycling lanes, etc. The huge amount of information provided by OpenStreetMap allows also to take into account speed limits and to know where points of interest are placed, which is useful to create realistically located traffic jams. OSMobility uses PostgreSQL[7] with PostGIS[8] extension to store OpenStreetMap data, and pgRouting[9] to generate routes. In practice, the OSMobility user has only to set endpoints, and routes are automatically generated. OSMObility's `Router` class, which provides an abstract method `generateRoute()`, has been specialized to `DijkstraRouter`, which uses Dijkstra's algorithm to compute the shortest route between two `OSMNodes`. In general, the best route may be defined according to any criteria.

One of the main features of OSMobility is the ability to define any mobility model, using the abstract class `MobilityModel` as a common base. We have specialized such a class to implement the Fluid Traffic Model (FTM) [49], where speed is a monotonic decreasing function of vehicle density. The FTM is used to compute the next position of a vehicle, given its current position, speed, direction, and the density of surrounding vehicles. Such a computation is performed by the `MoveNodeEvent` — whose instances are continuously generated and inserted in the event queue of the simulation engine, for every `MobileNode` that is moving (as illustrated in Figure 4.4).

The main advantage of using OSMobility is that it supports a wide range of mobility models, including both traditional ones (such as FTM) and custom, user-defined ones. Compared to *SUMO*[10] a well-known microscopic traffic simulator, OSMobility provides dynamic vehicle routing. SUMO does not natively include such a fea-

[7] *PostgreSQL* `https://www.postgresql.org/`

[8] *PostGIS* `http://postgis.net/`

[9] *pgRouting* `http://pgrouting.org/`

[10] *SUMO - Simulation of Urban MObility* `http://sumo-sim.org/`

ture, as routes have to be statically defined before the simulation is performed. Such a constraint can lead to unavoidable long trip times, or even to traffic blocks, for the low-priority roads which intersect high-priority ones. A number of frameworks exist to join SUMO and network simulators. Two notable examples are *Veins*[11] and *TraNS*[12] which respectively employ the OMNet++ and ns-2 network simulators. Both are based on the *TraCI* interface, which uses a TCP-based client/server architecture to allow the interaction with SUMO. Thereby, SUMO acts as a server and receives control messages to alter the simulator behavior at runtime.

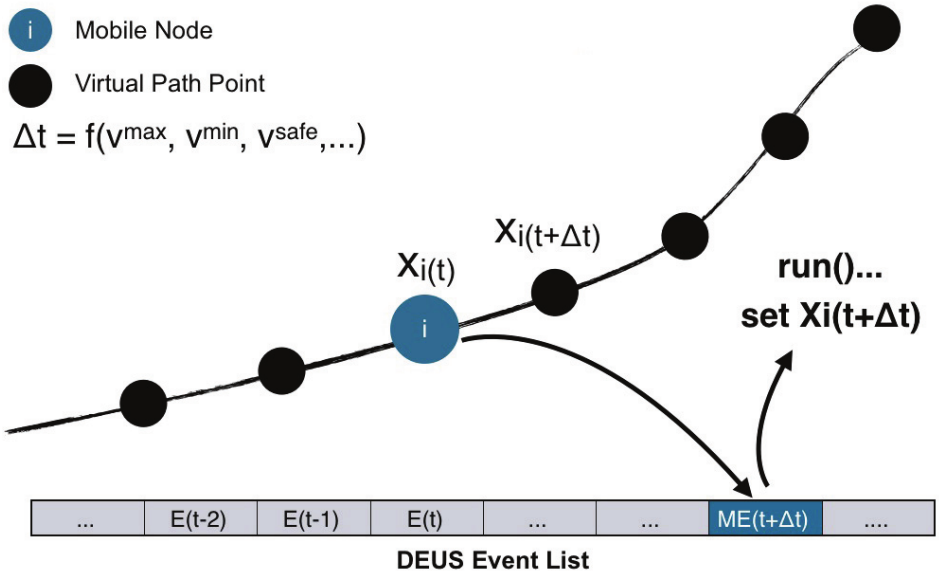


Figure 4.4: Sequence of `MoveNodeEvents` in the event queue of the simulation engine.

By adopting the Adapter pattern [50], we have designed our `IoTNode` as a wrapper of OSMobility's `GeoNode`. Thus, it is a node of the simulation and provided with a geographical location. It can be whether a stationary node or a mobile node. In the latter case, it can adopt any mobility model, keeping the high flexibility of OSMobil-

[11] *Veins - Vehicles in Network Simulation* `http://veins.car2x.org/`
[12] *TraNS* `http://www.isi.edu/nsnam/ns/`

ity.

4.2.4 Communication and Energy Packages

An `IoTNode`, in addition to a mobility model, has a network model and an energy model, represented by the generic classes `NetworkModel` and `EnergyModel`, respectively. By extending the `NetworkModel` class, it is possible to describe the capabilities of the node with regard to the network interfaces and protocols of communication. Indeed, developers can customize an `IoTNode` in order that it represents the required IoT device, with any network interface and protocol of communication. The `NetworkModel` has the task to calculate delays or even failures in the delivery procedure.

To simulate a distributed system with DEUS, it is necessary to write the classes that represent a message delivery from one node to another. In particular, it is necessary to define the sender, the destination and to schedule a *delivered message event* in the future (in terms of virtual time of the simulation). The scheduling time of such an event must be set using a suitable process, selected among those that are provided by the DEUS API, or defined by the user, possibly. The communication package provides several delay models which can be used to simulate message transmission between network nodes. The most simple model generates an exponential delay, whose expected value is computed from the message size and the nominal channel bandwidth. If the purpose of the simulation is to measure the average delay of propagating multi-hop messages within a network of nodes, the value of each link's delay must be realistic, taking into account the underlying networking infrastructure. In particular, if the communication is wireless, estimating the delay of point-to-point communication is a challenging task. Fortunately, this is not necessary, thanks to possibility to use dedicated simulation tools, such as Cooja and ns-3, to fully characterize the communication delay and packet losses.

Furthermore, the developer can implement the desired energy model, extending the `EnergyModel` class, so that it is possible to model in a highly sophisticated way, the battery consumption of the device. For example, it is possible to model the rate of the duty cycle for each `IoTNode`. Also in this case, Cooja can be used to

obtain a very detailed modeling of the energy consumption.

Cooja and ns-3

Cooja is a network simulator included in the Contiki system, which is an open source operating system for networked, memory-constrained systems with a particular focus on low-power wireless IoT devices. Contiki is often used in street lighting systems, sound monitoring for smart cities, radiation monitoring systems, and alarm systems. Cooja simulates networks of Contiki nodes, which may belong to either of three classes: emulated nodes, where the entire hardware of each node is emulated, Cooja nodes, where the Contiki code for the node is compiled for and executed on the simulation host, or Java nodes, where the behavior of the node must be reimplemented as a Java class.

Ns-3 is a discrete event network simulator for Internet systems. It is a free software project publicly available under the GNU GPLv2 license for research, development, and use. The ns-3 project is committed to building a solid simulation core that is well documented, easy to use and debug, and that caters to the needs of the entire simulation workflow, from simulation configuration to trace collection and analysis. Furthermore, the ns-3 software infrastructure encourages the development of simulation models which are sufficiently realistic to allow ns-3 to be used as a real-time network emulator, interconnected with the real world and which allows many existing real-world protocol implementations to be reused within ns-3. The ns-3 simulation core supports research on both IP and non-IP based networks. However, the large majority of its users focuses on wireless/IP simulations which involve models for Wi-Fi, WiMAX, or LTE for layers 1 and 2, and a variety of static or dynamic routing protocols such as Optimized Link State Routing Protocol (OLSR) and Ad hoc On-Demand Distance Vector (AODV) for IP-based applications.

Model Refinement

The direct integration of DEUS with Cooja and ns-3, with the first that “calls” the others to compute a delay value every time a node must send a message to another

node and the energy consumption, taking into account current surrounding conditions, is unpractical and would highly increase the simulation time. Instead, a more effective and efficient solution includes the following steps:

1. identify the main sub-system types, each one being characterized by specific networking features;

2. with Cooja or ns-3: create detailed simulation models of the sub-systems (*i.e.*, sub-models), and measure their characteristic transmission delays and the power profiling;

3. with DEUS: simulate the whole distributed system, with refined scheduling of communication events, taking into account the transmission delays and the realistic power consumption computed at step 2.

Amoretti *et al.* in [51] give a detailed explanation of the model refinement process from ns-3. A similar approach can be also adopted for Cooja.

4.2.5 Log Package

Deployment code running on a virtual node produces raw data (*e.g.*, received request rate, service rate, success rate), which are placed in the shared memory or in a database. Scheduled by the simulation engine, the logging package picks such raw data and produces aggregated logs in machine-readable format — ready to be analyzed and used to generate graphs. The logging package is highly generic, providing basic primitives for storing and retrieving data in shared memory / database. Its specialized use is defined by the adapter that must be developed, in order to integrate logging package and deployment code.

4.2.6 Performances of the Simulation Platform

To evaluate our simulation platform and, in particular, its scalability, we have decided to define several simulation scenarios and, for each of them, we have calculated the required time to complete the simulation.

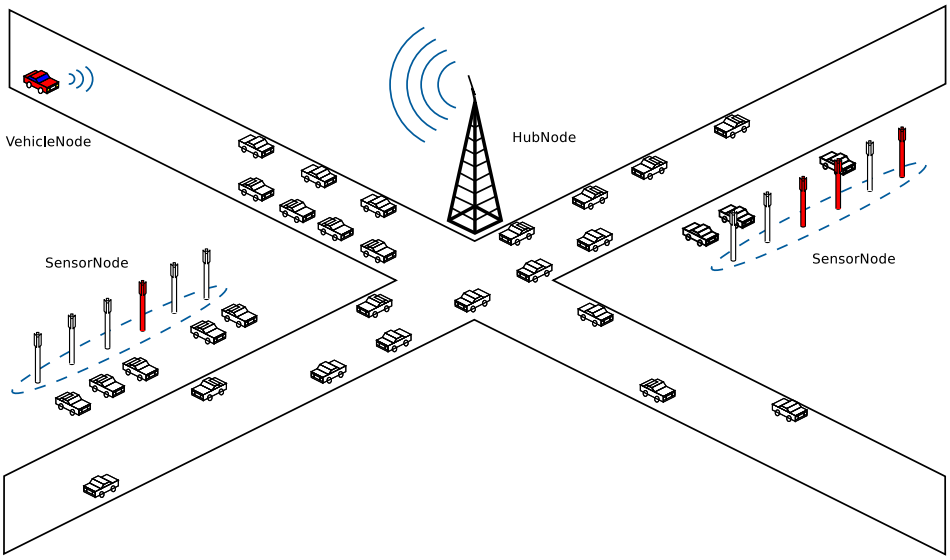


Figure 4.5: Sample IoT (smart-parking) scenario for the simulations.

The scenarios we have decided to simulate belong to a typical application case of the Internet of Things in urban environments: a smart parking infrastructure where in each parking lot a sensor node is deployed to detect the presence or absence of a vehicle [52]. Moreover, there are gateways that take charge of collecting data from the sensor network, and vehicles that move around the city and contact the gateways, searching for an available parking lot.

In particular, we have defined that vehicle requests to gateways are scheduled by a Homogeneous Poisson Process with interarrival time being an exponentially distributed random variable whose mean interarrival time is equal to 5 minutes. When a vehicle makes a request, it communicates with the geographically nearest gateway. The number of vehicles looking for a parking lot is chosen so that, at the end of the simulation, 10% of the traveling vehicles have issued a parking request to the gateways. Next, the gateway contacts the parking sensors under its own responsibility and waits for their response. Thereafter, it communicates a response to the vehicle. All this happens for a lifetime of the simulation equals to 4 hours. Figure 4.5 shows our

simulation scenario, where some vehicles travel around a generic city, send messages with gateways using LTE communication standard and also presents wireless sensors networks which transmit their data to the gateways.

All the participants of the simulation are implemented by subclassing our implementation of the `IoTNode` class. In particular, they are described by the `VehicleNode`, `HubNode` and `SensorNode` classes. In addition, they adopt the interface of mjCoAP[13], a well-known open source Java implementation of the CoAP protocol. In this way, code portability is even higher and the implementation can easily be transferred on real devices, such as, for example, those equipped with the new Java ME 8 platform [53].

In order to assess the scalability of our simulation platform, the number of nodes of the simulation has been varied, as reported in Table 4.1, and the required computation time has been measured. Simulations have been executed on a server with 2 GHz Xeon CPU, 16 GB RAM, Ubuntu GNU/Linux operating system.

	# `SensorNode`	# `HubNode`	# `VehicleNode`
A	4000	8	500
B	8000	16	1000
C	20000	40	2500
D	40000	80	5000
E	100000	200	12500
F	200000	400	25000

Table 4.1: Number of nodes of each simulation scenario.

Since the complexity of a simulation in a discrete-event simulation framework depends mostly on the number of events, rather than the number of nodes, for each simulation scenario, the number of scheduled events has been calculated, and reported in Table 4.2. It also presents also the percentage of mobility and communication events with respect to the total number of scheduled events. In particular, in our simulations

[13] *mjCoAP* `http://mjcoap.org`

we have adopted a very accurate mobility model and this entails the prevalence of mobility events rather than communication or birth events.

	# events	% mobility	% communication
A	2.7×10^6	98.5	1.4
B	5.6×10^6	98.5	1.3
C	1.4×10^7	98.5	1.3
D	2.7×10^7	98.4	1.4
E	6.3×10^7	98.3	1.5
F	1.2×10^8	98.2	1.6

Table 4.2: Number of events of each simulation scenario.

Figure 4.6 shows the results of simulations in terms of execution time. As the reader can see, the presented simulation platform can easily manage a very high number of nodes and the associated events. Considering that the SmartSantander testbed[14], which is one of the most famous real IoT platforms in urban environment, is composed of about 3000 IEEE 802.15.4 devices, 200 GPRS modules and 2000 joint RFID tag/QR code labels deployed both at static locations (streetlights, facades, bus stops) as well as on-board of mobile vehicles (buses, taxis), we can state that our simulation platform allows to easily simulate realistic scenarios in short time (scenarios B and C). Moreover, even with the enormous number of nodes we have considered in scenario F, the computation time is still acceptable, making the simulation platform particularly suitable for urban environment scenarios, where the number of nodes is often considerable.

4.3 Simulating the ADGT

The methodology described in previous section can be applied to the evaluation of different pervasive applications. Indeed, we illustrate the testing of a decentralized

[14] *SmartSantander* `http://www.smartsantander.eu/`

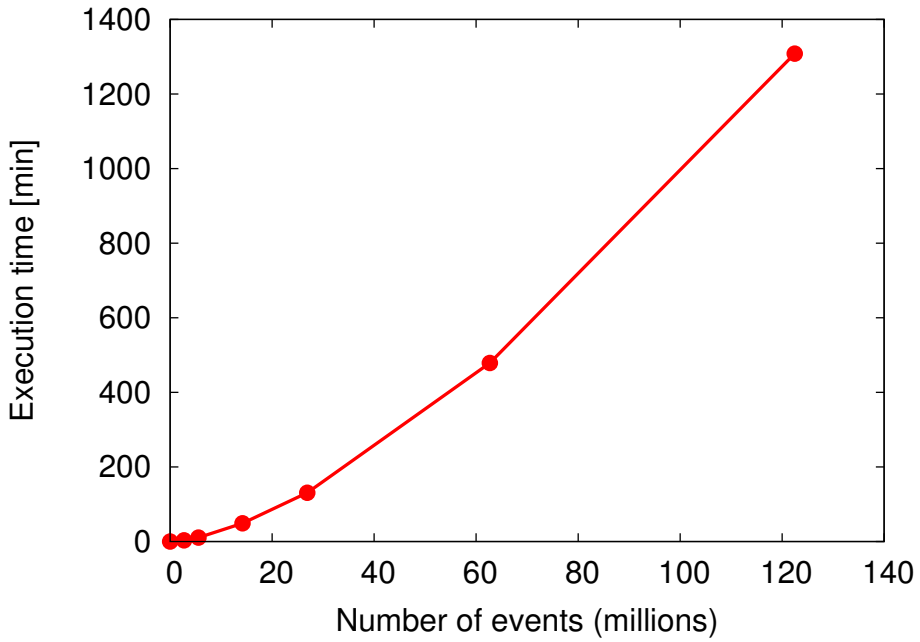


Figure 4.6: Execution time with respect to the number of events, for the six considered scenarios.

information sharing service based on ADGT, in the context of smart mobility. The related software can be installed on mobile devices, and used to produce, share and consume traffic information (such as accidents, traffic jams, detours). This is particularly useful for drivers, allowing to setup vehicular networks whose software is not embedded into vehicles — instead, it is portable from one vehicle to another, by the same user.

The software-in-the-loop simulation stack is illustrated in Figure 4.7. On top of DEUS, we have set the OSMobility, communication and logging packages. To integrate them with the mobile application, we have developed three lightweight adapters:

- the first one allows the mobile app to access position, way, route of the simulated node, generated by the OSMobility package;

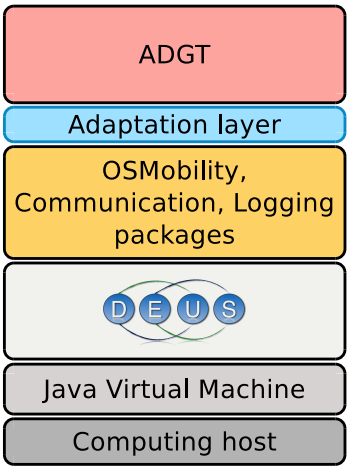


Figure 4.7: Layered representation of the testbed.

- the second one allows the mobile app to send/receive messages, by wrapping the communication package into a class which exposes the interface of a ADGT peer;
- the third one allows to pass mobile app state information (*e.g.*, number of sent messages) to the logging package.

Long-term statistics are periodically logged to file. For live monitoring, OSMobility provides a visualization module (a related screenshot is shown in Figure 4.8).

To simulate a network with 1000 vehicles, with high message rate (1.5 messages per second, each message being 200 B large) and high mobility resolution (1 simulation event corresponds to a 20 m displacement), a single node — 2 GHz Xeon CPU, 16 GB RAM — of our cluster is sufficient. The duration of such a detailed simulation takes 10 hours, on the average. Most of the time is devoted to the simulation of the message exchange between nodes. With the aforementioned hardware configuration, it is possible to simulate up to 4000 vehicles, in a reasonable time interval. For larger scenarios, there are two options: either using a more powerful server, or adopting a parallel approach [47].

Furthermore, the communication package exposes a Sip2Peer-like API, and frees

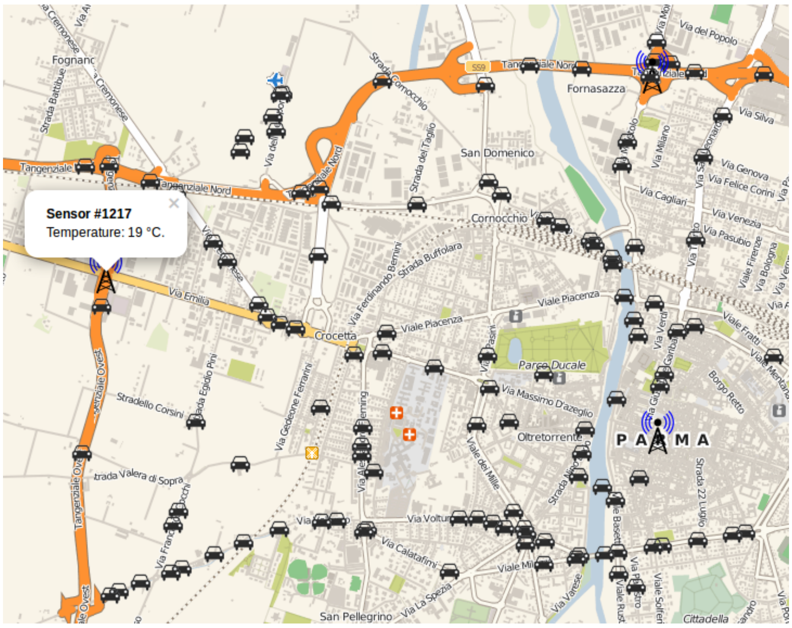


Figure 4.8: OSMobility visualization of networked vehicles.

developers from dealing with low-level communication issues, so that they can focus on the application-level functional requirements of the system of interest.

4.3.1 Mobility Model

The capability to define different models of mobility offered by OSMobility, allowed us to simulate the behavior of ADGT in three completely different LBS scenarios, where peers are vehicles that exchange messages:

1. vehicles moving within our university campus in the rush hour (Figure 4.91);

2. vehicles moving within our city (Parma), during ordinary traffic hours (Figure 4.92);

3. vehicles moving along the highways of our administrative region, *i.e.*, Emilia-Romagna (Figure 4.93).

(1) University campus (2) Parma (3) Highways in Emilia-Romagna

Figure 4.9: Scenarios adopted during simulations.

These scenarios differ not only in the number of peers, but also in the way they move. For example, within our university campus, in the rush hour vehicle speed is very low due to recurring traffic-jams. On the other hand, vehicles moving in Parma may run faster and travel along many different paths. Finally, most vehicles that travel along the highways do it around the speed limit and their direction is stable.

In all these scenarios, we have tried to choose configurations as much realistic as possible. In particular, in the simulations within our university campus, each peer randomly chooses two points of interest among the ones we have defined (such as the entrances or the sport center), and then follows the shortest path calculated between them. Also the number of vehicles is not randomly chosen, but is based on realistic assumptions. With regard to the simulations within Parma, vehicles randomly select two geographical locations and move along the fastest path that joins them. In the third scenario, peers move along the highways between two toll booths, still randomly chosen. Vehicles are simulated so that the traffic density is comparable to the actual one, as reported by the web portal about mobility of Emilia-Romagna[15].

In addition, every vehicle adopts the Fluid Traffic Model as mobility model, a realistic traffic model where the speed of vehicles is a monotonic decreasing function of the vehicle density [49].

[15] *E-R Mobilità* `http://mobilita.regione.emilia-romagna.it`

4.3.2 Communication Model

Since the main application of the ADGT we have conceived is based on the use of mobile devices like smartphones, the modeling of communication delays plays a fundamental role. For this reason, to better characterize the communication among the ADGT peers in the urban environment, we have adopted the model illustrated by Amoretti *et al.* [51], using ns-3 with the Lena LTE-EPC package[16]. The latter provides the E-UTRA part of the Long Term Evolution (LTE) technology, dealing with PRY, MAC and Scheduler functionalities, and support for the LTE RLC and PDCP protocol, together with EPC data plane features, such as the S1-U interface and the SGW and PGW entities. Shortly, such an ns-3 package supports the detailed simulation of end-to-end IP connectivity over LTE-EPC.

4.3.3 Configuration Parameters

In order to evaluate the behavior and the performance of the ADGT overlay scheme throughout the simulations, we have identified the main configuration parameters that characterize it. In particular, they are

- β: maximum number of peers returned by a discovery response;
- ε: minimum distance (dimension: [km]) that must be travelled by a peer, before it notifies its neighbors about its change of location;
- K: the number of GeoBuckets that constitute each GeoBucket structure;
- t: the thickness (dimension: [km]) of each GeoBucket;
- L: the maximum number of peers contained in a GeoBucket;
- S: the value which relates the speed of the peer to the shape of the GeoBucket.

[16] *LENA - LTE-EPC Network Simulator* `http://networks.cttc.es/mobile-networks/software-tools/lena/`

4.3.4 Evaluation Metrics

Both quantitative and qualitative performance evaluation metrics have been taken into account. Quantitative metrics show the cost of the ADGT in terms of transmitted data — this is an important aspect, since typically smartphones, and in general mobile devices, have traffic tariffs based on transmitted data amount. On the other hand, qualitative metrics show the behavior of the ADGT from a point of view of the quality of the service. So, they represent capabilities of the overlay scheme such as the warning responsiveness or the service accessibility.

In particular, we have defined

- Data Rate (DR), calculated as the average number of bits that are transmitted per unit of time by each peer (dimension: [kbit/s/peer]);
- Coverage Percentage (CP), calculated as the number of peers that have actually received a specific warning message, over those that should have received it;
- Distance From Events (DFE), calculated as the average distance between the geographical location of peers which have not received yet a warning message and the geographical location of the associated event.

In order to compute CP and DFE values, a randomly chosen peer periodically produces a warning message related to its geographical location and disseminates it among its neighbors.

4.4 Results

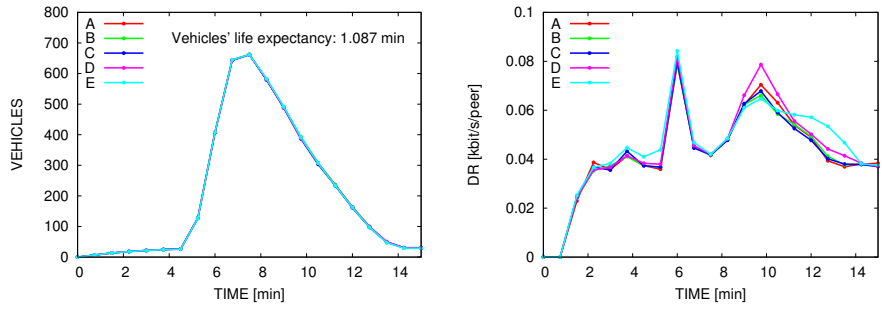
Taking into account our past experiments with the DGT, we have decided to set $\beta = 20$, $\varepsilon = 1$, $K = 5$, $t = 0.4$, $L = 20$. Since the most characteristic parameter of the ADGT is S, we have defined five different configurations, denoted by the letters A, B, C, D and E, corresponding to increasing S values (as listed in Table 4.3).

The first simulation scenario has been defined to test the response of the ADGT in a situation of sudden high traffic density. We have modeled what typically happens

	A	B	C	D	E
S	0	1.5	2.5	5	10

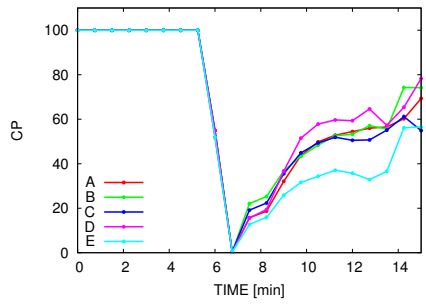
Table 4.3: Configuration parameters of the simulations.

in our university campus at the end of the day, *i.e.*, a significant number of vehicles leave the parking areas and move towards the points of interest.

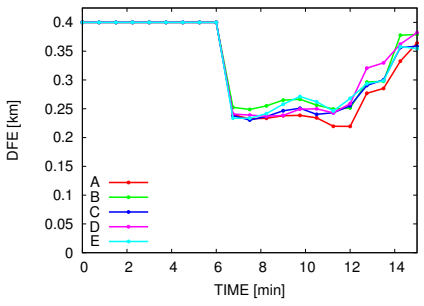


(a) Number of Vehicles

(b) Data Rate



(c) Coverage Percentage



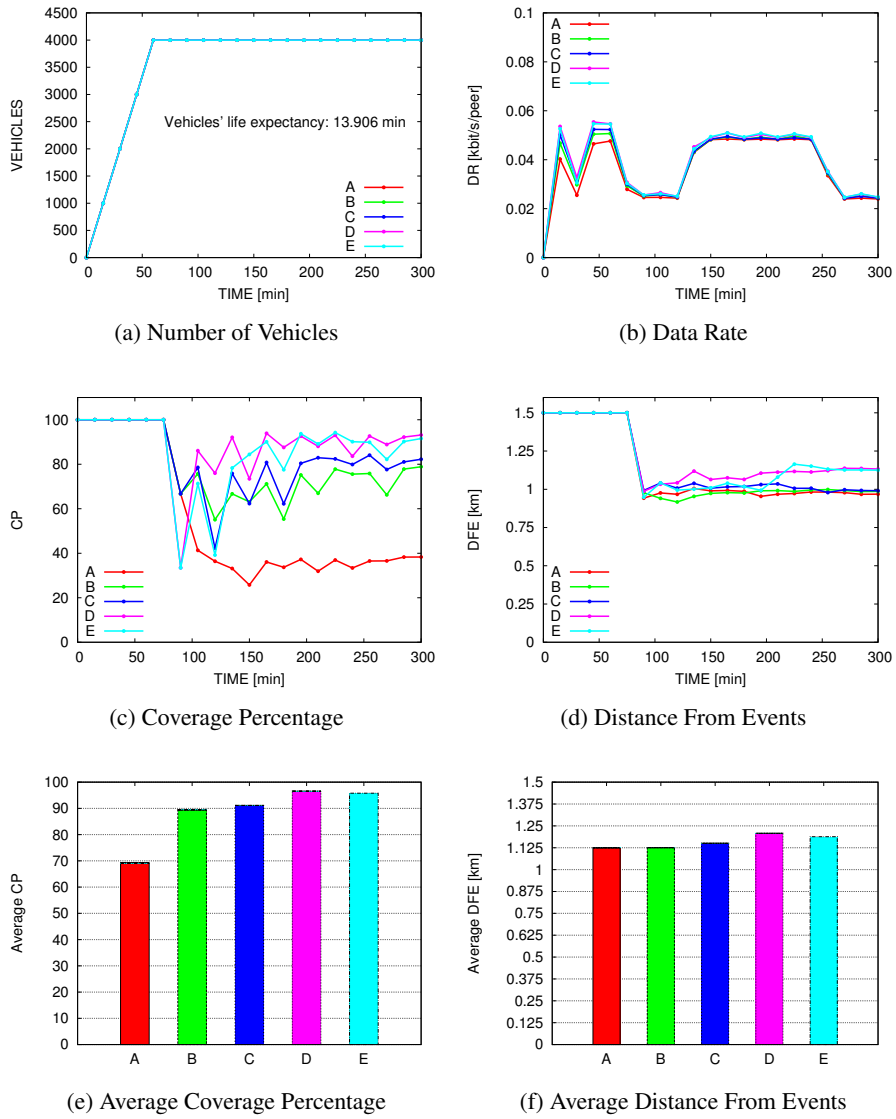
(d) Distance From Events

Figure 4.10: Simulation results of the scenario inside the university campus.

Simulation results presented in Figure 4.10 show that a too large value of S badly affects the qualitative performance (CP and DFE values) of the ADGT, because of the very small area of the university campus. The best S value, among those con-

sidered, is the one related to configuration D. Figure 4.10a shows the trend of the number of vehicles during the simulation. The warning message has been generated from a single randomly chosen peer, after 6 minutes from the beginning of the simulation. Figure 4.10b shows the DR of each peer. As the reader can see, the DR grows very slowly with the progress of the simulation, which is a very good result since it confirms the possibility to employ the ADGT on real mobile devices. Figure 4.10c shows the CP of the generated warning message, which grows from a very low value (when the warning message has been generated) to a high value, after a few minutes. Finally, one of the most interesting results is shown in Figure 4.10d, where the average of the DFE starts from a value of 0.2 km and rapidly arrives at the value of 0.4 km, which is the distance within the warning message has validity. In summary, the most noticeable result of this simulation is that a good quality of service can be guaranteed, despite the impossibility to set up a stable network topology, because of the very short routes the vehicles cover in the university campus.

On the other hand, the second scenario allows to evaluate the behavior of the ADGT in a situation of normal traffic condition. In particular, vehicles move on the roads of Parma, covering a period of five hours. During the first simulated hour, a vehicle is generated every 0.9 seconds. Every vehicle is removed, once it reaches its destination, and substituted by a new vehicle, placed at a randomly chosen location. In this way, after the first hour of simulation, there are constantly 4000 traveling vehicles. In addition, starting from the first hour and with a period of 30 minutes, a randomly chosen vehicle generates a warning message associated to the vehicle's geographical location, with a validity distance equals to 1.5 km. Figure 4.11 illustrates the simulation results of this second scenario. In particular, in Figure 4.11a the reader can see the number of vehicles. Figure 4.11b shows that also in this simulation the DR is very low. Compared with the evaluation of the DGT in a similar urban scenario, the data rate is reduced by one order of magnitude [13]. This remarkable result is mainly due to the recursive routing algorithm used by the ADGT. Figure 4.11c shows that the ADGT is highly responsive, considering that, if the generation of a new event implies a natural decrease of the CP value, this is quickly followed by a fast growth. In this case it is possible to observe another important result: the gap between the CP



(a) Number of Vehicles

(b) Data Rate

(c) Coverage Percentage

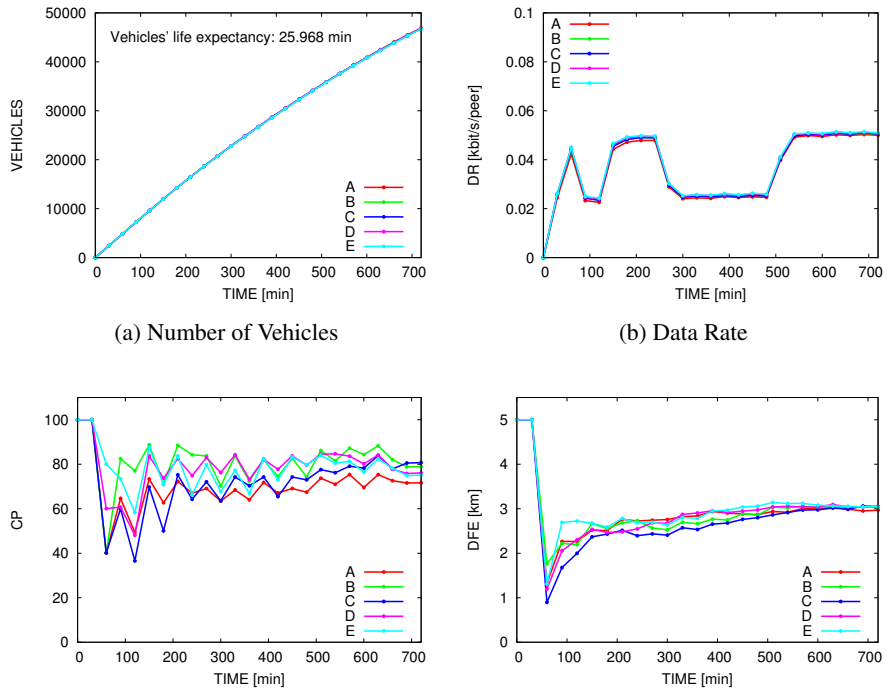
(d) Distance From Events

(e) Average Coverage Percentage

(f) Average Distance From Events

Figure 4.11: Simulation results of the scenario inside the city of Parma.

values of simulation A (where $S = 0$) and the others. This demonstrates that an adaptive overlay, *i.e.*, with $S > 0$, is much more efficient than a static overlay. Also the



(a) Number of Vehicles

(b) Data Rate

(c) Coverage Percentage

(d) Distance From Events

Figure 4.12: Simulation results of the scenario along the highways.

DFE trend in Figure 4.11d, confirms this feature, since, after each warning message generation, it remains almost constant. Figures 4.11e and 4.11f show more clearly that the performance quality increases from configuration A to configuration D. In particular, with respect to configuration A, configuration D provides a CP increment of about 27 percentage points. There is no clear advantage in using configuration E instead of D.

Finally, the last simulated scenario allows to evaluate the ADGT in a highly structured environment, where the direction of the peers does not change frequently. In particular, we have considered the highway segments that connect three cities in the Emilia-Romagna region, namely Piacenza, Parma and Reggio nell'Emilia. The

simulation spans 12 hours and adopts an arrival period of the vehicles at the toll booths equals to 0.74 seconds. When a vehicle is generated, it travels towards another randomly chosen toll booth. Once the destination has been reached, the vehicle is removed from the simulation and not replaced. Furthermore, every hour a random vehicle generates a warning message with a range of validity equals to 5 km.

The results of this simulation are shown in Figure 4.12. In Figure 4.12a, the reader can see that the simulation ends before a constant number of vehicles is reached, due to the (realistic) mobility model parameters we have used. The Data Rate is presented in Figure 4.12b. Again, it is very low, compared with the current traffic data rates available to mobile devices. Finally, Figure 4.12c and Figure 4.12d show that, also in this scenario, the ADGT demonstrates a very good efficiency in the transmission of information.

Chapter 5

Evaluation through Field Testing

In this chapter, a concrete LBS example based on Adgt.js is presented, in order to illustrate how straightforward and powerful such a framework is. Furthermore, by means of this, we conducted field testing in real environment with the aim of obtaining a performance evaluation of the framework. At the end of the chapter, results of the test are reported and compared to those obtained in simulations.

5.1 Implementing LBSs with Adgt.js

Designing and implementing LBSs with Adgt.js is straightforward, being not too different from realizing a modern web page. Moreover, being Adgt.js an implementation of a peer-to-peer protocol, it is practically affordable by whoever, as it may run over any type of device, given it is not particularly demanding in terms of computing and memory resources.

In order to use the Adgt.js framework, it is sufficient to include the JavaScript file in the HTML page using the `src` attribute in the `<script>` tag. Once the Adgt.js is included, it is already possible to create a new ADGT peer, as shown in Listing 5.1.

```
1 var peer = new Peer(options);
2 peer.connect();
```

Listing 5.1: Creating a new ADGT peer.

Only two statements are required for adding a new peer to the overlay: the first one actually creates an instance of the `Peer` class, while the second one starts the connection to the ADGT overlay network. During the creation of the peer, it is possible to specify some options, such as the address of the bootstrapping node, the STUN and TURN servers, as well as some parameters of the ADGT protocol.

Adgt.js has been developed with an event-driven approach, thus allowing the definition of functions that are executed upon the occurrence of certain events, such as the reception of a message from another peer, or a change in the neighborhood.

In Listing 5.2, it is reported how to set listener functions for two kinds of events: `neighbors` and `data`. The first event occurs when the neighborhood of the peer changes, while the second one fires when the peer receives any type of data from near peers. The callback functions allow us to manage the set of neighbor `descriptors` and received `data`, respectively.

```
1 peer.on('neighbors', function(descriptors) {
2     ...
3 });
4 peer.on('data', function(descriptor, data) {
5     ...
6 });
```

Listing 5.2: Setting listeners for peer events.

To geocast `data` to the geographic neighbors of the peer, it is sufficient to invoke the `geocast` method, whereas to directly transmit data to a specific neighbor, one can use the `send` method, as in Listing 5.3.

```
1 peer.send(data, descriptor);
2 ...
3 peer.geocast(data);
```

Listing 5.3: Sending data to neighbors.

In case the geographic location of the peer changes, it is sufficient to use the `move` method to automatically trigger the position update process, that involves the transmission of a message to the neighbors and the removal of those peers no longer to be included in the GeoBucket (Listing 5.4).

```
1 peer.move(position);
```

Listing 5.4: Updating peer's position.

We have implemented and published online[1] a LBS that illustrates the ease of use of Adgt.js and represents a building block for more sophisticated applications. The LBS is a peer-to-peer instant messaging application that allows a peer to exchange georeferenced messages among its neighbors. In particular, the application shows two tabs that allow to switch from two different views:

1. a map that displays the peers connected to the network, *i.e.*, visitors of the web page, that are at a maximum distance of 40 km;

2. a chat with the georeferenced messages received from neighbors.

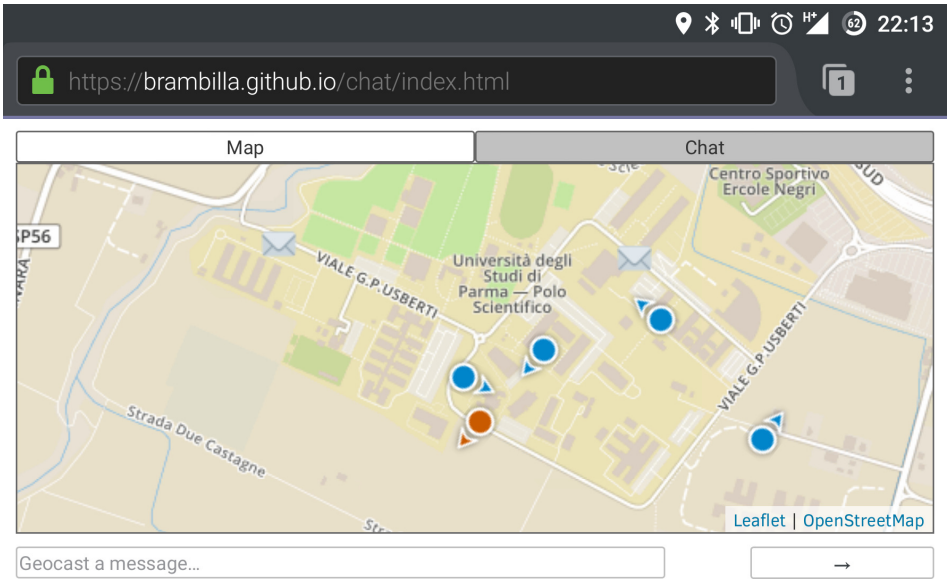


Figure 5.1: The LBS developed with Adgt.js, running on Firefox for Android.

[1] *LBS* `https://brambilla.github.io/chat/index.html`

In the LBS, neighbor discovery and connection establishment is entrusted to Adgt.js. Regarding the map, we have adopted Leaflet[2], a widely used open source JavaScript library used to build web mapping applications.

Figure 5.1 is a screenshot taken from an Android smartphone running the LBS on Firefox for mobile, where a red marker represents the geographical position of the peer in execution on the Android smartphone, and blue markers stand for geographical neighbors. Moreover, there are some envelope-shape icons that indicated georeferenced messaged created by the peers of the ADGT overlay network. To read the messages, the user can tap these icons or, alternatively, can switch to the tab specific for the chat.

This LBS only requires to update the geographic location of the peer and change the marker on the map with the position of the browser obtained with the Geolocation API, as shown in Listing 5.5.

```
1  navigator.geolocation.watchPosition(function(position) {
2    peer.move(position);
3    var latLng = L.latLng(position.coords.latitude, position.coords.longitude);
4    marker.setLatLng(latLng);
5    if (position.coords.speed > 0) {
6        marker.setIcon(icon_heading);
7    }
8    marker.setRotationAngle(position.coords.heading);
9    map.panTo(latLng);
10 });
```

Listing 5.5: Managing current position of the device.

In addition, when a change in the neighborhood happens, all the markers on the map representing neighbors are updated as in Listing 5.6.

[2] *Leaflet* `http://leafletjs.com`

```
peer.on("neighbors", function(descriptors) {
  markers.clearLayers();
  for (var index in descriptors) {
    var position = descriptors[index].position;
    var latLng = L.latLng(position.coords.latitude, position.coords.longitude);
    var rotationAngle = position.coords.heading;
    if (position.coords.speed > 0) {
      markers.addLayer(L.marker(latLng, {rotationAngle: rotationAngle, icon: icon_heading}));
    } else {
      markers.addLayer(L.marker(latLng, {rotationAngle: rotationAngle, icon: icon}));
    }
  }
});
```

Listing 5.6: Managing changes of neighborhood.

5.2 Field Experiments

Starting from results obtained through simulations, we have arranged a series of field tests on the Adgt.js implementation, thanks to the kind cooperation of two groups of students that have evaluated the implemented LBS described above, using their own smartphones connected to Internet through cellular connectivity. In particular, these students have used the LBS to exchange messages while they were moving around inside our university campus. Due to the way in which such students, *i.e.* the peers, moved and to the largeness of the area available to moving peers, this experiment loosely recalls the second simulated scenario, *i.e.*, the one where peers were vehicles moving through ordinary traffic in Parma.

The first group consisted of about 15 students. Figure 5.2 shows the evolution of connected peers, while Figure 5.3 shows the number of georeferenced messages exchanged by them. As shown in Figure 5.2, after about 10 minutes from the start of the evaluation, the majority of the users are connected and, from that moment, the number of georeferenced messages rises steeply (Figure 5.3).

To effectively evaluate the performances of the Adgt.js, comparing it with the ADGT overlay scheme tested in simulations, we decided to extract the same information of the simulations and, in particular, we have obtained the same indicators: CP and DFE. In order to do this, we have added a tracking mechanism to the LBS so that, using the Websocket protocol, communicates to a server all the significant

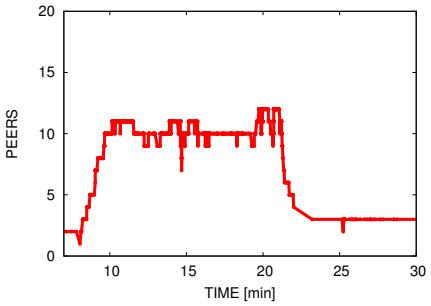


Figure 5.2: The number of peers connected to the LBS (first experiment).

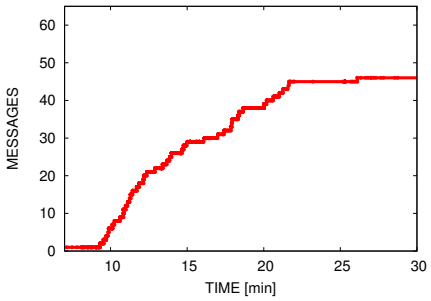


Figure 5.3: The number of messages exchanged by peers (first experiment).

events, *i.e.* messages received, neighbors known, connection and disconnection from the network, and so on. In this way, once the experimentation was completed, we have analyzed the information acquired from the server and calculated the CP and DFE indicators.

In particular, Figure 5.4 shows the CP of messages transmitted by users to their geographical neighbors during the experimentation. As illustrated in Figure 5.4, the evolution of CP in the experiment with the Adgt.js implementation is not particularly different from that in the simulations, despite the behavior of peers in the former is not as well structured as in the latter, since in simulations the peers join regularly to the network and practically without haphazardness. Moreover, in the experimentation with Adgt.js, where real users are involved, the exchange of messages starts even if the network is not completely and fully connected differently from the simulations and this entails a greater difficulty in spreading messages. Furthermore, it is important to take into account the fact that in the simulations there are not potential connectivity issues nor heterogeneity in the adopted mobile telecommunications technology, therefore messages are always transmitted in the same way. Instead, the field testing has been carried out with the personal smartphones of students and, consequently, with different mobile network operators, operating systems, hardware features and ultimately different capabilities that affect the entire peer-to-peer network. Nevertheless, Figure 5.4 shows that the value of CP remains high enough to ensure proper

operation of the ADGT overlay network.

Figure 5.5 depicts the trend of DFE of messages exchanged by users of the LBS implemented with the Adgt.js framework. Also in this case results are quite satisfying since, despite the area of interest is larger than that of simulations having the messages a validity distance equals 4 km, the minimum value of DFE is high enough to allow the peers to receive in advance the messages.

Results presented in Figure 5.4 and Figure 5.5 show that the behavior of Adgt.js is consistent with what we obtained with simulations. Therefore, field experiments and simulations validate reciprocally.

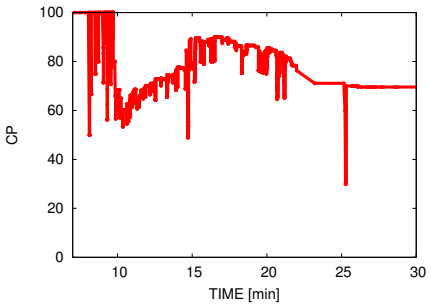


Figure 5.4: The Coverage Percentage of messages exchanged by peers (first experiment).

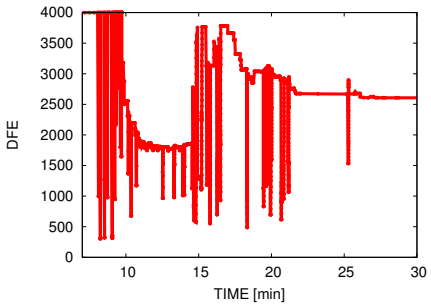


Figure 5.5: The Distance from Events of messages exchanged by peers (first experiment).

Later, we organized another test session involving a larger group of students (about 50). In this case, owing to the number of peers with respect to the size of the area in which they moved, and due to the sudden boost of peers connected to the peer-to-peer network, this experiment somehow recalls the first simulated scenario, the one in which vehicles move during rush hour.

As a matter of fact, about a 22% of the students was connected for less than 1 minute, while another 26% was connected for more than 1 minute and less than 2 minutes. Moreover, about a 38% of the students was connected to the ADGT overlay network for more than 2 minutes and less than 5 minutes, while the remaining students were connected for more than 5 minutes.

With these test conditions, results are highly positive, as illustrated in Figures 5.6, 5.7, 5.8 and 5.9.

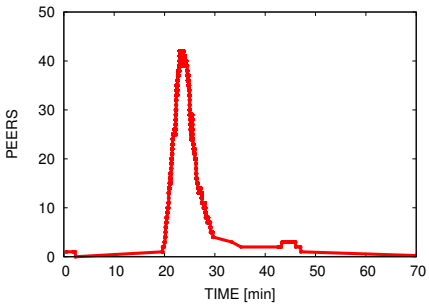


Figure 5.6: The number of peers connected to the LBS (second experiment).

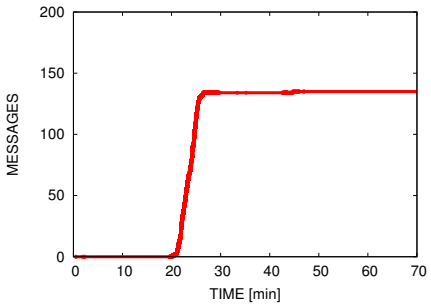


Figure 5.7: The number of messages exchanged by peers (second experiment).

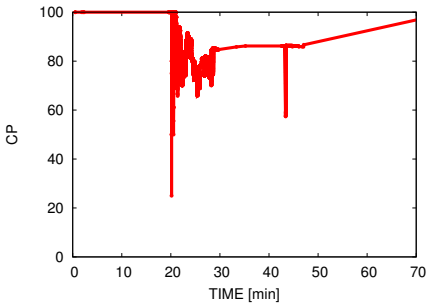


Figure 5.8: The Coverage Percentage of messages exchanged by peers (second experiment).

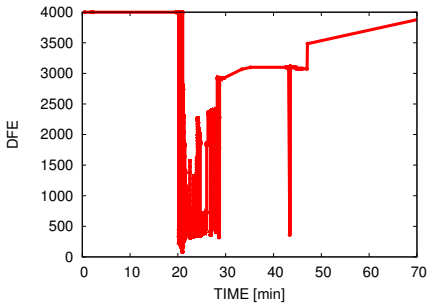


Figure 5.9: The Distance from Events of messages exchanged by peers (second experiment).

Chapter 6

Blockchain for Proof-of-Location

In recent years, an increasing number of location-based services have been released, mostly because of the rapid expansion of the mobile device market. LBSs take advantage of geographic location to provide users with accurate and targeted information for locating friends on a map, discovering nearby social events, crowdsensing applications such as generating alerts about traffic jams along a route, and more.

To ensure that such services work properly, it is necessary that geographic locations claimed by users are factual. For example, LBSs with location-based access control that allow users to obtain a discount coupon, require that users cannot cheat on their position, to avoid delivering coupons to those who really did not deserve them. Similarly, social networks that enable users to discover where their friends are, work correctly only if geographic locations are certified.

In this chapter, it is presented a novel approach for guaranteeing location trustworthiness and preserving user privacy, at the same time. In particular, we have realized a completely decentralized mechanism for the validation of the geographical position, suitable for location-based peer-to-peer overlay schemes, such as the ADGT [54].

6.1 Related Work

The term *proof-of-location* refers to a method by which a system can confirm the geographic locations of users, *i.e.*, a digital certificate that attests someone's presence at a certain geographic location, at some point in time, whereby LBSs can validate user locations. In literature, different approaches to this topic have been investigated and proposed. Lenders *et al.* [55] presented a trusted computing module that ensures GPS data generated and transmitted by mobile devices. This solution is not acceptable inasmuch the trusted computer module would be expensive and difficult to adopt. Saroiu and Wolman [56] proposed a solution where users prove their geographic locations through a Wi-Fi infrastructure. Not even this approach is affordable, as it relies on a complex PKI infrastructure that needs to be deployed and maintained. SMILE [57] utilizes wireless techniques to prove if a physical encounter occurred. However, this service does not reveal the actual location information to the service provider thus can only provide location proofs between two users who have actually encountered. Another solution introduced by Zhu and Cao [58] envisages a network infrastructure that provides proof-of-location of mobile devices equipped with Bluetooth. Although this solution appears effective and robust, its centralized architecture eases tracking of pseudonym-identified users by malicious administrators, whereas it might hinder the deployment of user-created location-based services.

Starting from the ideas of Zhu and Cao, we have designed a completely distributed protocol that provides proof-of-location, while preserving privacy of users. Indeed, the decentralized nature of peer-to-peer systems guarantees higher privacy levels, as it removes the central authority knowing both the geographic location of users and the information they exchange. To endow location-based peer-to-peer overlay schemes with proof-of-location, while preserving the decentralized approach, we based our protocol on the *blockchain*, which is mostly known for being Bitcoin's core technology [59].

In the following, peer-to-peer network, overlay network, peer-to-peer overlay and network are equivalent expressions we use with reference to the same concept.

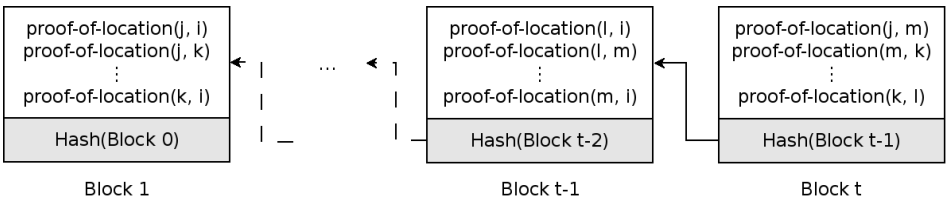


Figure 6.1: Proofs-of-location recorded in blocks of a blockchain. Every block contains a hash of the previous block.

6.2 Blockchain Approach for Proof-of-Location

The blockchain technology is a novel peer-to-peer approach, which allows to maintain a continuously-growing list of data records, linked in a way that makes them immutable. In general, a block is a set of one or more data records, prefaced by a block header and protected by a proof of some type. The initial and most widely known application of this technology is Bitcoin's public transaction ledger, a digital asset and a payment system invented by Satoshi Nakamoto [59].

The main feature that differentiates the blockchain from all other distributed databases is its completely decentralized nature, which escapes the presence of a trusted central authority. Indeed, blockchain maintenance is performed by a network of communicating nodes, which store their own copy of the blockchain, validate transactions, add them to their copy of the blockchain, and then broadcast block additions to other nodes. All these operations are performed in such a way that *consensus* emerges among network nodes, about the information stored in the blockchain.

A blockchain database consists of two kinds of records: transactions and blocks. Transactions are authenticated by mass collaboration powered by collective self-interests. The result is a robust workflow where participants' uncertainty regarding data security is marginal. Blocks hold batches of valid transactions that are hashed and encoded into a Merkle tree. Each block includes the hash of the prior block in the blockchain, linking the two. The linked blocks form a chain. This iterative process confirms the integrity of the previous block, all the way back to the original genesis block.

We have adopted the blockchain technology to endow networked nodes with the capability to verify and store geographic locations, not requiring a centralized super-node that oversees sensitive data of other nodes. In our approach, recent valid proofs-of-location are recorded into blocks, which are then added to the end of the chain and, once confirmed by consensus, they cannot be changed, as shown in Figure 6.1.

In particular, each peer i of the network is described by a unique id K_i^{pu}, which is also its *public key*. Moreover, every peer i is able to digitally sign messages with the *private key* K_i^{pr} associated with its id.

6.2.1 Blockchain Construction

Similarly to the solution proposed by Zhu and Cao [58], peers can communicate with near nodes through any short-range communication technology, such as Bluetooth, Bluetooth SMART or ZigBee, and they periodically use these interfaces to broadcast proof-of-location requests and responses to their neighbors.

Supposing next block to be confirmed is $Block_t$, a proof-of-location request from peer i to peer j contains the identifier of peer i (*i.e.*, K_i^{pu}), the geographic location of peer i, and a hash of the preceding block in the blockchain $h(Block_{t-1})$. The request is signed with the requester's private key, so that anyone can verify that it has not been tampered with, as depicted in Figure 6.2.

$$Req_{i \to j} : \left\{ \begin{array}{c} K_i^{pu} \\ \langle latitude, longitude \rangle_i \\ h(Block_{t-1}) \\ timestamp \end{array} \right\}_{K_i^{pr}}$$

Figure 6.2: A proof-of-location request produced and signed by the peer i for the peer j.

The peer that receives the proof-of-location request (*i.e.*, peer j) verifies its validity, according to the following rules:

1. the request has to come from a peer that, beyond being in touch through the short-range communication technology, is a known contact in the location-based peer-to-peer overlay scheme;
2. the request is digitally signed by the peer that produced it;
3. the request contains an admissible geographic location, *i.e.*, not further than the adopted maximum distance reachable with the short-range communication technology;
4. the request refers to the end block of the blockchain.

Once all checks have been fulfilled, a proof-of-location response is produced. The responding peer wraps the received request in a new message, together with its geographic location and identifier (*i.e.*, its public key K_j^{pu}). The proof-of-location response is also signed with the private key of the responding peer, as illustrated in Figure 6.3.

$$Res_{j \to i} : \left\{ \begin{array}{c} Req_{i \to j} \\ K_j^{pu} \\ \langle latitude, longitude \rangle_j \\ timestamp \end{array} \right\}_{K_j^{pr}}$$

Figure 6.3: A proof-of-location response produced and signed by peer j for peer i.

The response is verified in a similar way to the request:

1. the response comes from one of the peers to whom the request was sent;
2. the response is digitally signed by the peer that produced it;
3. the response contains an admissible geographic location, *i.e.*, not further than the maximum distance that is reachable with the adopted short-range communication technology.

In case the response is correctly verified, it corresponds to a proof-of-location, attesting that two peers are geographically close to each other and specifying their geographic locations. Proofs-of-location are then broadcast to the network, which records them in the public record of all proofs-of-location, namely the blockchain, after validating them. When a peer receives proofs-of-location related to peers that should be located nearby, it checks for their presence in the list of contacts provided by the peer-to-peer overlay. It is expected that geographic locations specified in proofs-of-location are reasonably close to each other, within the limits of the adopted short-range communication technology. If these constraints are not fulfilled, proofs-of-location are discarded and not disseminated within the network.

Every peer in the network puts all known valid unacknowledged proofs-of-location into a block, together with a reference to the previous valid block known to that peer. In addition to proofs-of-location and the reference to its predecessor, the block contains the identifier of the peer that generated it. Moreover, the block is signed with the private key of the peer that generated it, as shown in Figure 6.4.

$$Block_t : \left\{ \begin{array}{c} Res_{j \to i} \\ Res_{j \to k} \\ \vdots \\ Res_{k \to i} \\ K_i^{pu} \\ h(Block_{t-1}) \end{array} \right\}_{K_i^{pr}}$$

Figure 6.4: t-th block, produced and signed by the peer i.

Afterwards, the newly created block is broadcast to the peers of the network, which decide whether to add the block to the end of the blockchain or not. If the majority of peers adds the block to the blockchain then consensus is achieved, therefore proofs-of-location are made persistent. Otherwise, the block is discarded and not attached to the blockchain.

Whereupon, it is verified that the hash of the referenced block matches the end block in the chain, otherwise a *fork* in the blockchain occurs.

Which one branch will become part of the main blockchain depends on the distributed consensus algorithm explained below. Afterwards, the peer makes sure that proofs-of-location specified in the block are not already present in previous blocks of the blockchain. In case a proof-of-location concerns the geographic location of the peer itself, it is checked that signatory peers of the proof-of-location are known (*i.e.*, they belong to the contact list provided by the peer-to-peer overlay). If these conditions are not respected, the block is discarded, instead of being propagated into the network.

6.2.2 Distributed Consensus

Unlike Bitcoin, where distributed consensus is achieved by means of a proof-of-work technique, in our approach, a *proof-of-stake* mechanism is adopted, whereby next valid block in the blockchain is the one produced (*mined*) by the peer that owns the majority of proofs-of-location in the latest T blocks of the blockchain. In Bitcoin, peers are encouraged by the reward they earn to repeatedly run hashing algorithms to validate transactions. Conversely, our protocol does not require such an extremely time-consuming work, thus there is no reward for mined blocks, albeit it can be provided at the application layer. Therefore, in case a peer receives more than one valid block from its neighbors, it will add to the end of its blockchain the one produced by the peer that has received the largest number of proofs-of-location, in the latest T blocks. Moreover, to prevent the monopoly problem, *i.e.*, a peer that keeps out proofs-of-location that concern other peers from the block it produces with the purpose to remain the peer that owns the majority of proofs-of-location and, therefore, that takes control of the blockchain, the protocol prevents that the same peer generates more than one block in the latest T blocks of the chain. This way of defining the next valid block in the blockchain is inspired by Peercoin's proof-of-stake system [60] that is based on the concept of *coin age*, a number derived from the product of the number of coins times the number of days the coins have been held.

Remarkably, as distributed consensus is obtained according to information con-

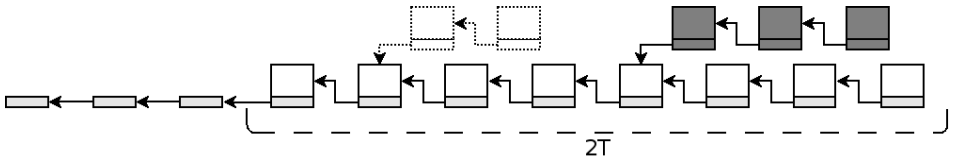


Figure 6.5: Valid proofs-of-location are persisted in the main blockchain (white blocks). Proofs-of-location in the latest $2T$ blocks are no longer significant. Grey blocks compose a fork competing to become the main blockchain. Dashed blocks are part of a past fork that has become invalid and not used for anything.

tained in the latest T blocks of the chain ($2T$ to handle forks of the blockchain), it is not necessary that every peer of the network handle all blocks of the blockchain. The protocol works properly even if peers do not consider proofs-of-location contained in the blocks that precede the latest $2T$, but only the identifier of the peer that has produced it and the reference to the previous block, as depicted in Figure 6.5.

The value of T depends on the application layer. When it is important to store many past geographic locations, such as in applications for tracking and monitor vehicle fleets, T has a higher value, compared to applications that localize nearby friends. Forensic applications may be interested to store the whole blockchain, in order to provide effective and trusted alibis for people under investigation. On the other hand, the lower is the value of T, the smaller is also the space occupied in memory. Therefore, the protocol is independent from the application layer and even versatile for the realization of different LBS types.

6.3 Protocol Analysis

We have analyzed the behavior of the proposed protocol, in case it is exposed to attacks that affect proof-of-location mechanisms, thus resulting to be particularly dangerous for LBSs [61].

1. **Cheating on own geographic location.** A peer could attempt to counterfeit its geographic location, specified in a proof-of-location request or response, in

order to obtain a proof-of-location attesting that its geographic location is different from the actual one. Our protocol prevents this kind of attack, since each peer that receives a proof location request or response, verifies that the specified geographic location is not further than the maximum distance reachable with the adopted short-range communication technology.

2. **Cheating on another peer's geographic location.** Another possible attack could hail from a peer that produces false claims about other peers' geographic locations. The protocol precludes such an attack, thanks to the asymmetric cryptography mechanism, whereby all the declarations concerning geographic locations stated by peers are digitally signed with their private keys and easily verifiable using their public keys that correspond to their identifiers.

3. **Replaying proofs-of-location.** Proofs-of-location could be rebroadcast in the network by a malicious peer, with the purpose to forge its geographic location or that of other peers. Since every peer of the network checks that the proof-of-location is not already contained inside the blockchain before retransmitting it, it is not possible to successfully complete this attack. Moreover, inasmuch every proof-of-location contains a reference to a block of the blockchain, it is immediately discarded in case the referenced block is older that the latest T blocks of the blockchain.

4. **Colluding with other peers.** Another threat exists when two or more peers collude with each other to generate counterfeit proofs-of-location. In literature, this kind of attack is denoted as *Sybil attack* and it happens when a malicious peer tries to prove itself in a geographic location that is not the actual one, with the help of another peer. Indeed, two peers could agree upon producing a proof-of-location attesting that their geographic locations are different from real ones, and broadcast it in the network.

   Since our protocol relies on a location-based peer-to-peer protocol where peers expect to be directly connected, in the overlay network, with their geographic neighbors, colluding peers can be easily identified by means of the short-

range communication technology. Moreover, it is unlikely that the whole list of neighbors provided by the peer-to-peer protocol is made of colluding peers. For the sake of precision, there are four possible situations:

(a) **Proof-of-location and location declared in the peer-to-peer overlay are equal and both counterfeit.** If a peer receives a proof-of-location concerning two other peers that claim to be close to it, it verifies that at least one of the two peers can be contacted with the short-range communication technology; if not, the proof-of-location is discarded.

(b) **Proof-of-location and location declared in the peer-to-peer overlay are different and both counterfeit.** If a peer receives from another peer a proof-of-location concerning the latter peer and related to a geographic location that is different from the one provided by the peer-to-peer overlay, such a proof-of-location is immediately discarded.

(c) **Proof-of-location is counterfeit; the location declared in the peer-to-peer overlay is real.** This situation is addressed in the same way of the previous one.

(d) **Proof-of-location is real; the location declared in the peer-to-peer overlay is counterfeit.** Also this case, which is probably the most intuitive, is resolved like the second one.

Hence, collusion is hindered by information provided by peers belonging to the location-based peer-to-peer overlay.

5. **Determining real identity of peers.** An attacker could attempt to determine the real identity of peers through full observation of proofs-of-location in the blockchain. Actually, in our protocol there is no limit on the number of identifiers: in the same way as Bitcoin protocol allows the use of more than one receiving address, users of our protocol can freely decide to change their peer identifiers. As proved by Zhu and Cao [58], if a peer has the possibility to periodically change its identifier according to a Poisson distribution, it gains

unobservability and an attacker cannot determine the real identity of the peer by observing location proof records.

Thus, our protocol is shown to be robust against all major LBS-related attacks.

6.4 Protocol Evaluation

Thanks to the versatility of OSMobility, it has been possible to simulate the behavior of peers of a network based on our blockchain-based protocol for proof-of-location, and evaluate the algorithm that lead to the consensus among peers.

In particular, we have conducted different simulations with 400 peers within an area comparable to the surface of our university campus. The simulated time interval is one hour. During this period, peers generate and exchange proof-of-location every 15 minutes.

We have supposed that peers are connected using the ADGT protocol, where GeoBuckets have a radius of 2 km (see Section 2.2 of Chapter 2) and the Coverage Percentage is 75%, which is in line with all that described in Section 4.4 of Chapter 4. Moreover, the radius of the simulated short-range communication technology is set to 150 m.

The simulations we have carried out differ in the value of two parameters: the percentage of peers that generate proofs-of-location ($P1$), and the percentage of cheater peers ($P2$). To be more precise, the former is a parameter that specifies the number of peers that generate and broadcast a proof-of-location with the information of their geographical location and the location of another peer randomly chosen among peers within the radius of the short-range communication; the latter represents the number of peers of the network that cheat, *i.e.*, do not communicate their real geographical location but a counterfeit one. To simulate the Sybil attack, we have ensured that cheater peers always manage to collude with another peer, therefore they always generate a proof-of-location with a counterfeit geographical location. In Table 6.1, all the combinations of the $P1$ and $P2$ values are illustrated.

In Figures 6.6 to 6.17 four measured quantities are reported for all the considered configurations:

	A	B	C	D
P1 (%)	2.5	2.5	2.5	2.5
P2 (%)	0	25	50	100
	E	**F**	**G**	**H**
P1 (%)	5	5	5	5
P2 (%)	0	25	50	100
	I	**L**	**M**	**N**
P1 (%)	10	10	10	10
P2 (%)	0	25	50	100

Table 6.1: Configuration parameters of the simulations.

1. *True PoL*, the number of real (true) proofs-of-location generated by peers;
2. *False PoL*, the number of counterfeit (false) proofs-of-location generated by cheater peers;
3. *Accepted True PoL*, the number of real (true) proofs-of-location that, by consensus, are mined in the blockchain;
4. *Accepted False PoL*, the number of counterfeit (false) proofs-of-location that, by consensus, are mined in the blockchain.

In particular, plotted results come from 5 executions of each simulation, using different random seeds. In this way, the confidence interval is satisfactory since for all these quantities on average the standard deviation is not higher than 1 proof-of-location.

All plots clearly show that the algorithm of distributed consensus gives excellent results with respect to real proofs-of-location, since almost all of them are accepted by the peers, therefore mined in the blockchain. Indeed, the evolutions of *True PoL* and *Accepted True PoL* are overlapping for the most part.

Regarding the counterfeit proofs-of-location, results are still good, inasmuch most of them are discarded by peers. Moreover, as shown by the figures, the capability to

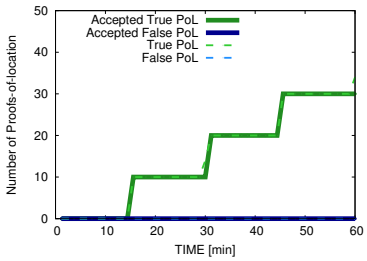


Figure 6.6: Results in A configuration ($P1 = 2.5\%, P2 = 0\%$).

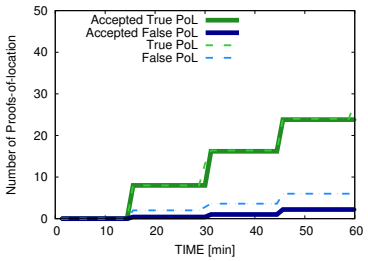


Figure 6.7: Results in B configuration ($P1 = 2.5\%, P2 = 25\%$).

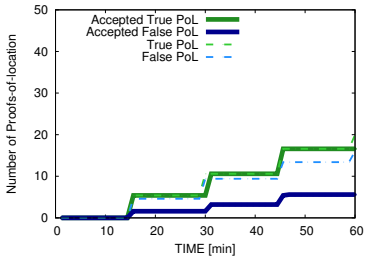


Figure 6.8: Results in C configuration ($P1 = 2.5\%, P2 = 50\%$).

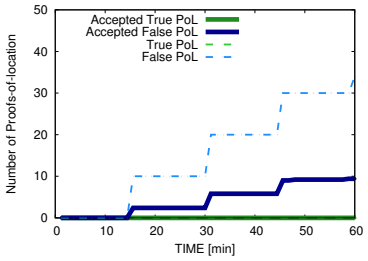


Figure 6.9: Results in D configuration ($P1 = 2.5\%, P2 = 100\%$).

distinguish between false and true proofs-of-location is not affected by the number of cheater peers: increasing the number of false proofs-of-location, it does not involve an increase in the number of the accepted false proofs-of-location.

Furthermore, the algorithm that determined the distributed consensus works well even if the percentage of generated proofs-of-location grows exponentially.

To better represent the good behavior of the algorithm, we have calculated the average accuracy in each simulation. The accuracy has been calculated using the formula

$$Accuracy = \frac{\sum True\ positive + \sum True\ negative}{Total\ population}$$

where

$$\sum True\ positive = Accepted\ True\ PoL$$

$$\sum True\ negative = False\ PoL - Accepted\ False\ PoL$$

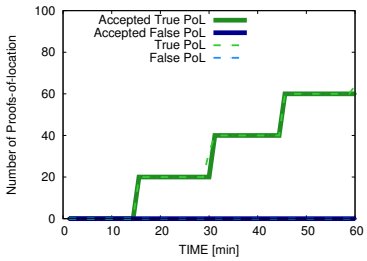


Figure 6.10: Results in E configuration ($P1 = 5\%, P2 = 0\%$).

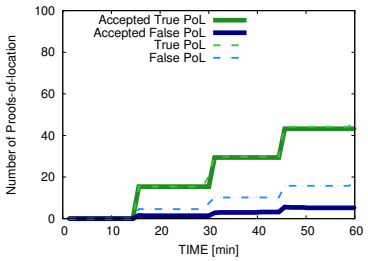


Figure 6.11: Results in F configuration ($P1 = 5\%, P2 = 25\%$).

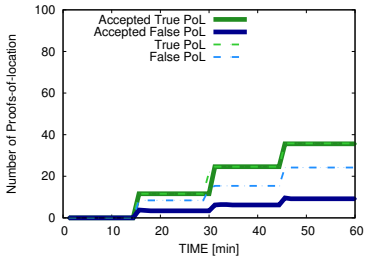


Figure 6.12: Results in G configuration ($P1 = 5\%, P2 = 50\%$).

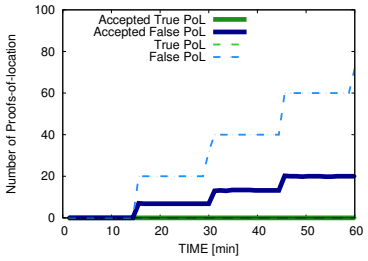


Figure 6.13: Results in H configuration ($P1 = 5\%, P2 = 100\%$).

$$Total\ population = True\ PoL + False\ PoL$$

.

As shown in Figure 6.18, the increase of cheater peers minimally conditions the value of accuracy, that remains always on high values. In particular, the histogram depicted in Figure 6.18 shows that the number of peers that generate proofs-of-location barely affects the average accuracy; indeed, the tendency of the histogram is essentially the same, in the three groups of simulations identified with the three different values of parameter $P1$ ($P1 = 2.5\%$, $P1 = 5\%$ and $P1 = 10\%$).

These simulation results confirm plainly that the algorithm works correctly, especially considering that the scenarios taken into account are extremely pessimistic, as it is unlikely that half of peers is cheater.

In order to try out the algorithm in even more critical conditions, we decided to

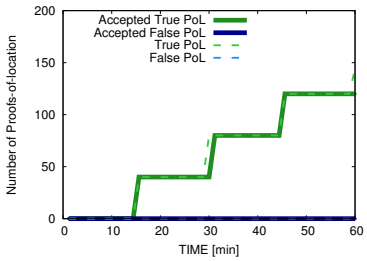


Figure 6.14: Results in I configuration ($P1 = 10\%, P2 = 0\%$).

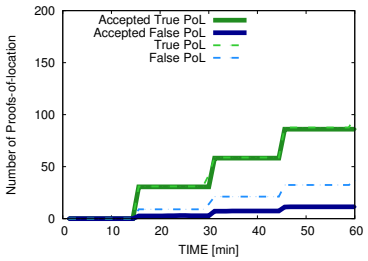


Figure 6.15: Results in L configuration ($P1 = 10\%, P2 = 25\%$).

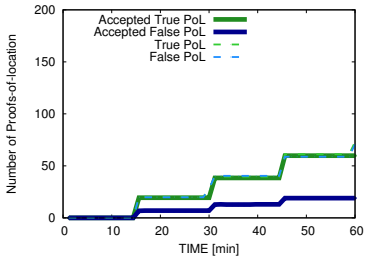


Figure 6.16: Results in M configuration ($P1 = 10\%, P2 = 50\%$).

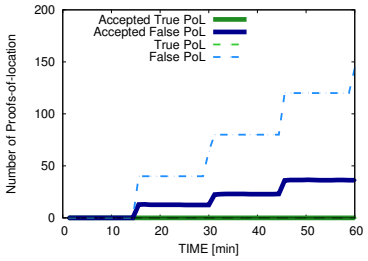


Figure 6.17: Results in N configuration ($P1 = 10\%, P2 = 100\%$).

repeat all the previous simulations, in an geographical area comparable to the surface of Parma and with a thousand peers. By doing so, the density of peers per unit area changes from about 226 $peers/km^2$ to 12 $peers/km^2$, therefore the peer-to-peer overlay has fewer interconnections among its peers.

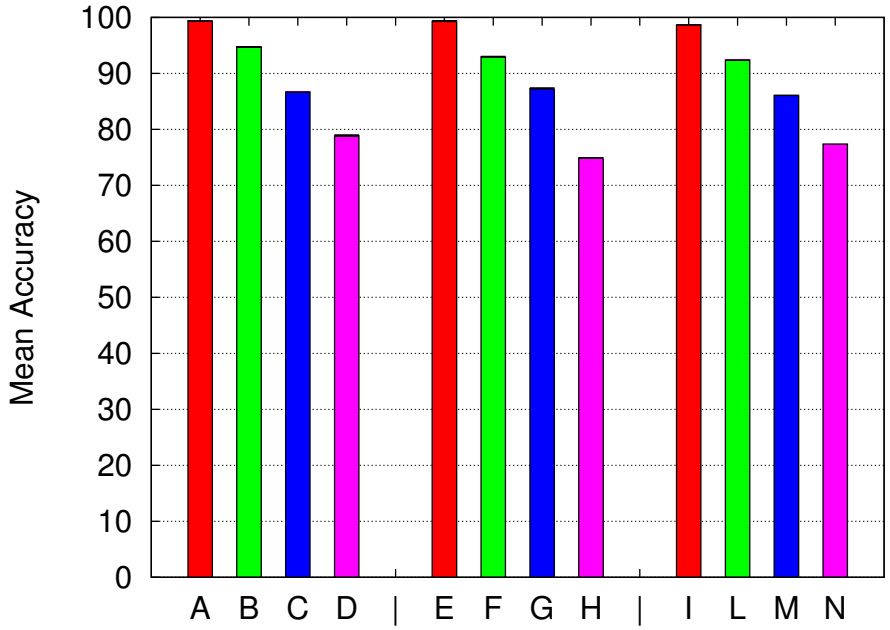


Figure 6.18: Mean accuracy in all simulations.

In Figures 6.19 to 6.30 are reported the four measured quantities for the simulations with less density of peers, with all the same configurations as before.

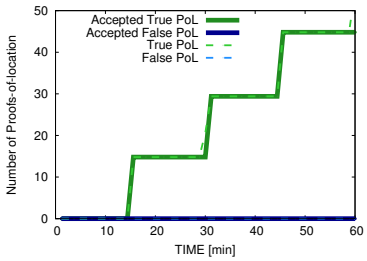


Figure 6.19: Results in A configuration with low density of peers ($P1 = 2.5\%, P2 = 0\%$).

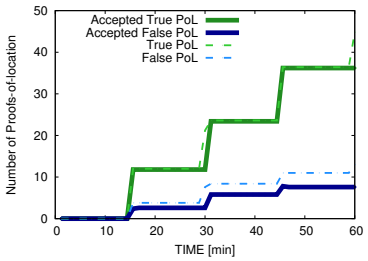


Figure 6.20: Results in B configuration with low density of peers ($P1 = 2.5\%, P2 = 25\%$).

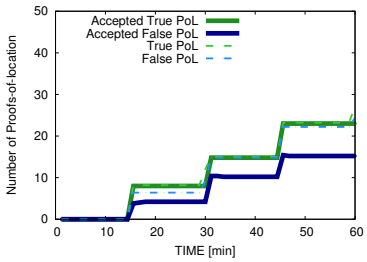


Figure 6.21: Results in C configuration with low density of peers ($P1 = 2.5\%, P2 = 50\%$).

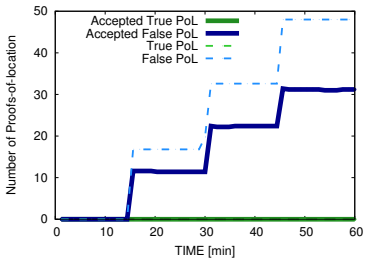


Figure 6.22: Results in D configuration with low density of peers ($P1 = 2.5\%, P2 = 100\%$).

Figures 6.19 to 6.30 show that the algorithm of distributed consensus continues to give excellent results with respect to real proofs-of-location. As regards the counterfeit proofs-of-location, the results are certainly worse than previous simulations, but this depends exclusively on the fact that the peer density is very low. Indeed, in these simulations peers are so scattered and distant that it is difficult to reject false proofs-of-location. On the other hand, the greater is the density of peers, the higher is the probability that a false proof-of-location is recognized as such by peers in the immediate vicinity.

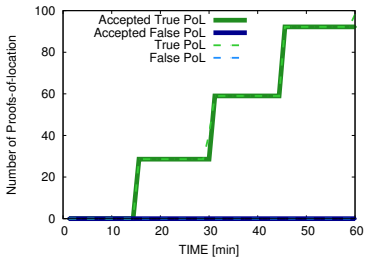


Figure 6.23: Results in E configuration with low density of peers ($P1 = 5\%, P2 = 0\%$).

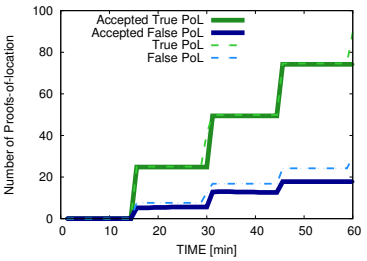


Figure 6.24: Results in F configuration with low density of peers ($P1 = 5\%, P2 = 25\%$).

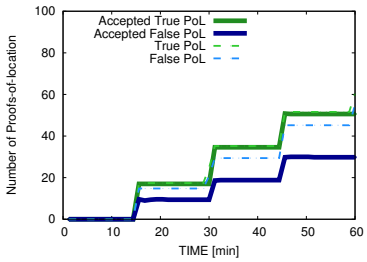


Figure 6.25: Results in G configuration with low density of peers ($P1 = 5\%, P2 = 50\%$).

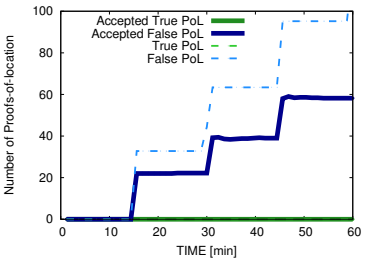


Figure 6.26: Results in H configuration with low density of peers ($P1 = 5\%, P2 = 100\%$).

Figure 6.31 shows the average of accuracy of the algorithm in each simulation. Although, as mentioned, the performance deteriorates, the behavior maintains the same positive trend.

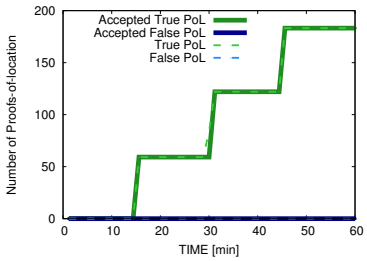


Figure 6.27: Results in I configuration with low density of peers ($P1 = 10\%, P2 = 0\%$).

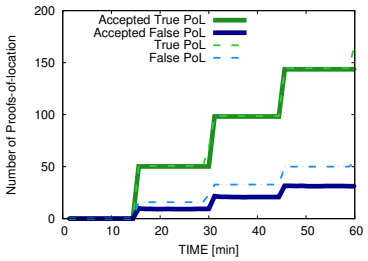


Figure 6.28: Results in L configuration with low density of peers ($P1 = 10\%, P2 = 25\%$).

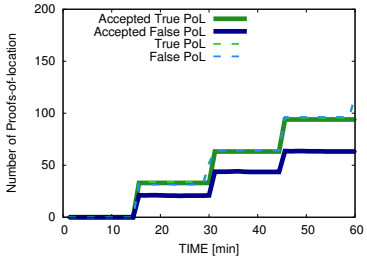


Figure 6.29: Results in M configuration with low density of peers ($P1 = 10\%, P2 = 50\%$).

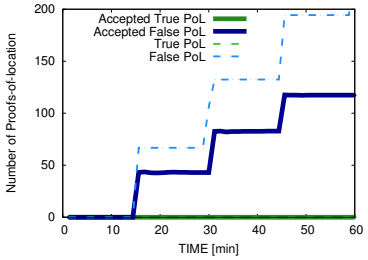


Figure 6.30: Results in N configuration with low density of peers ($P1 = 10\%, P2 = 100\%$).

Finally, we repeated the first simulations with 400 peers, changing the stochastic process that describes the model with which peers generate proofs-of-location. In particular, we switched from a periodic process with a period of 15 minutes to a Poisson process with an interrarrival (interoccurence times) of 15 minutes. In this way, we obtained more realistic scenarios of simulations.

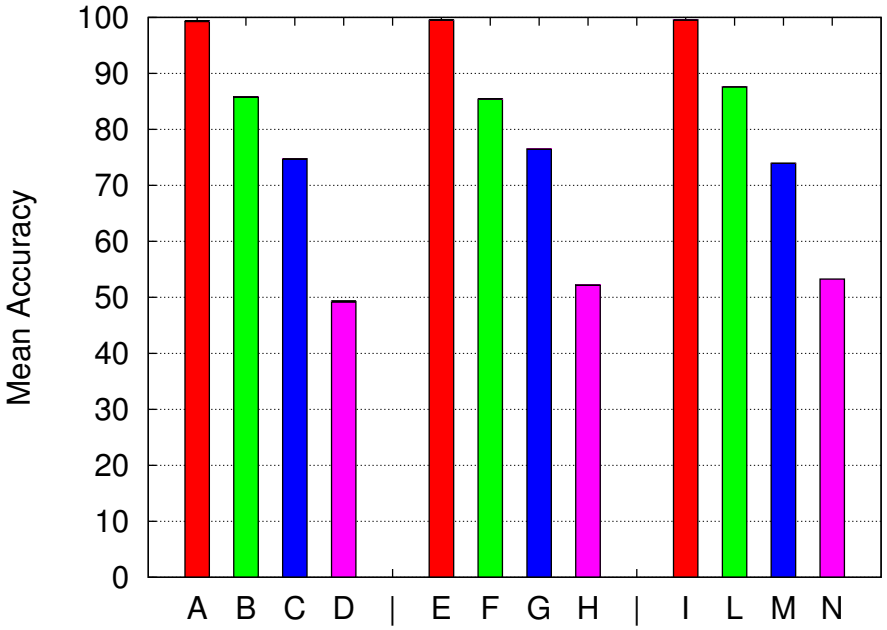


Figure 6.31: Mean accuracy in all simulations with low density of peers.

In Figures 6.32 to 6.43 are reported the four measured quantities for these simulations, with all the same configurations as before.

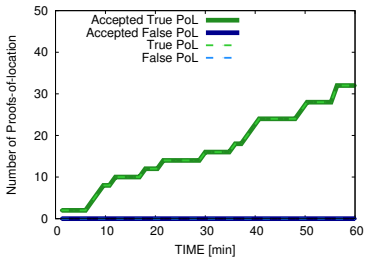


Figure 6.32: Results in A configuration with a realistic generation model of proofs-of-location ($P1 = 2.5\%, P2 = 0\%$).

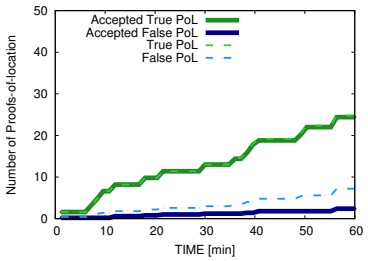


Figure 6.33: Results in B configuration with a realistic generation model of proofs-of-location ($P1 = 2.5\%, P2 = 25\%$).

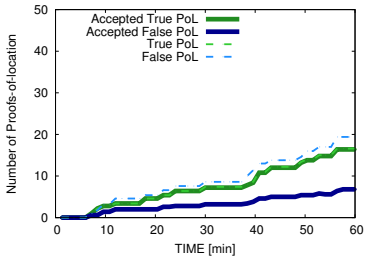


Figure 6.34: Results in C configuration with a realistic generation model of proofs-of-location ($P1 = 2.5\%, P2 = 50\%$).

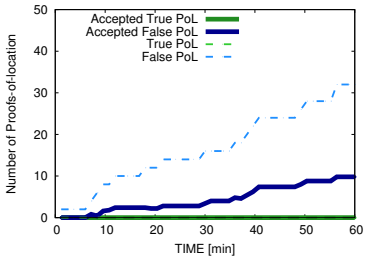


Figure 6.35: Results in D configuration with a realistic generation model of proofs-of-location ($P1 = 2.5\%, P2 = 100\%$).

Figure 6.44 shows that the average of accuracy of the algorithm in each simulation with a realistic generation model of proofs-of-location maintains very positive values.

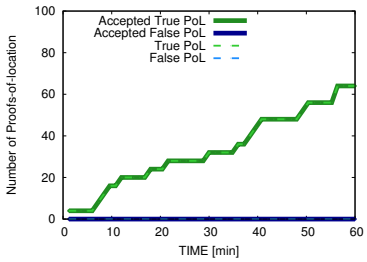


Figure 6.36: Results in E configuration with a realistic generation model of proofs-of-location ($P1 = 5\%, P2 = 0\%$).

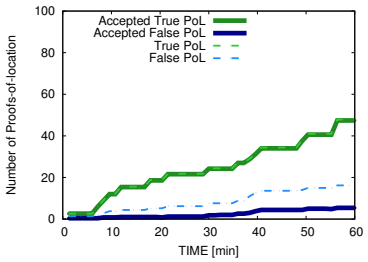


Figure 6.37: Results in F configuration with a realistic generation model of proofs-of-location ($P1 = 5\%, P2 = 25\%$).

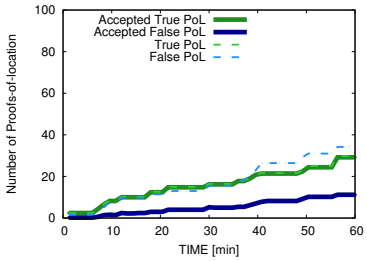


Figure 6.38: Results in G configuration with a realistic generation model of proofs-of-location ($P1 = 5\%, P2 = 50\%$).

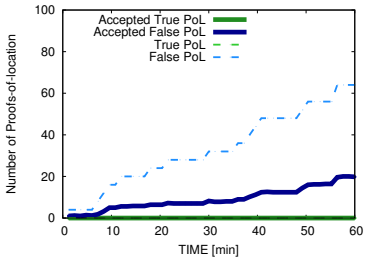


Figure 6.39: Results in H configuration with a realistic generation model of proofs-of-location ($P1 = 5\%, P2 = 100\%$).

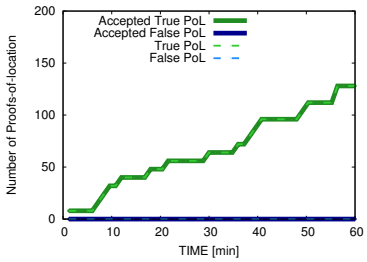


Figure 6.40: Results in I configuration with a realistic generation model of proofs-of-location ($P1 = 10\%, P2 = 0\%$).

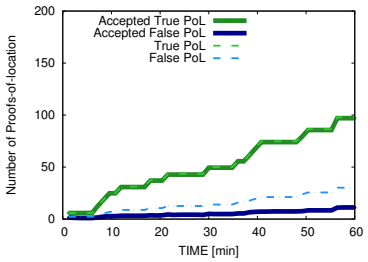


Figure 6.41: Results in L configuration with a realistic generation model of proofs-of-location ($P1 = 10\%, P2 = 25\%$).

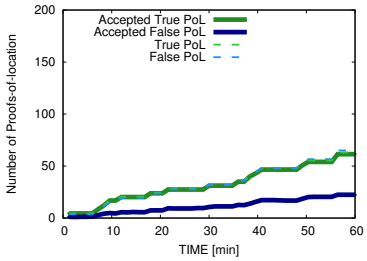


Figure 6.42: Results in M configuration with a realistic generation model of proofs-of-location ($P1 = 10\%, P2 = 50\%$).

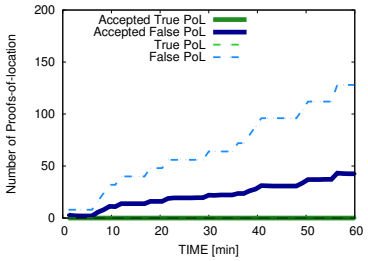


Figure 6.43: Results in N configuration with a realistic generation model of proofs-of-location ($P1 = 10\%, P2 = 100\%$).

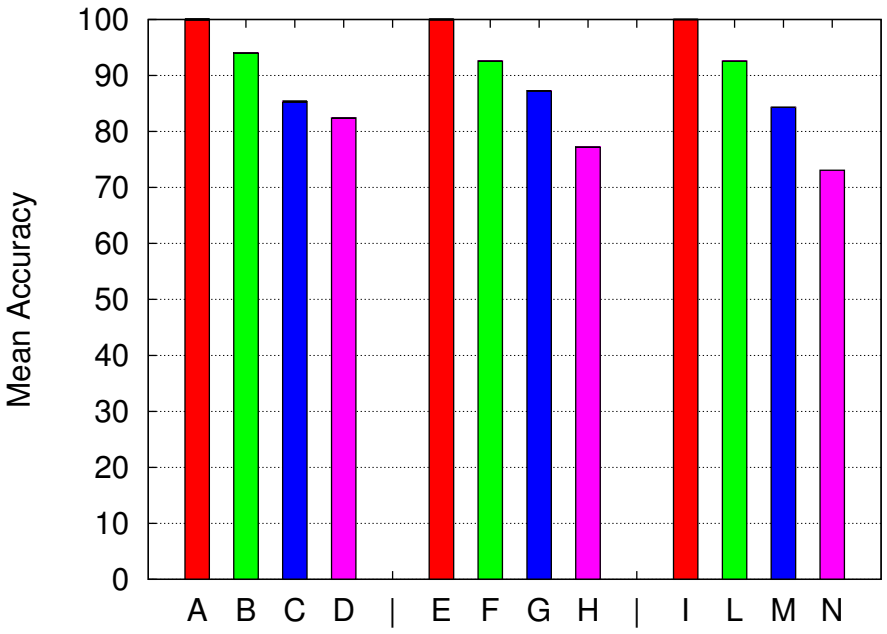


Figure 6.44: Mean accuracy in all simulations with a realistic generation model of proofs-of-location.

Chapter 7

Conclusions

In this Thesis, we have presented the Adaptive Distributed Geographic Table (ADGT) a novel adaptive peer-to-peer overlay scheme that allows the realization of location-based services for mobile peers, such as georeferenced information subscription and retrieval, as well as location-specific message broadcasting.

To the best of our knowledge, the ADGT is the first peer-to-peer overlay scheme that effectively takes into account the speed and direction of the peers to adapt its topology. Moreover, with respect to the state of the art, the ADGT provides much more location-based functionalities, such as a publish/subscribe mechanism through which peers can be notified about any information they are interested in around any geographical location, as well as geocasting functionalities.

Before the ADGT was introduced, peer-to-peer overlay schemes for location-based services existed, but none of them really considered the mobility of peers to enhance the overlay itself and to make more efficient the interconnections among peers of the network. During this Ph.D. activity, starting from general principles of one of the existing peer-to-peer protocols, we have defined a new mechanism for peers discovery and an innovative network topology that is able to adapt to peers' movements.

Furthermore, differently from other solutions that have been studied only in simulative environments, we have realized a truly usable implementations of the afore-

mentioned peer-to-peer overlay scheme. Indeed, the Adgt.js is a web application framework that enables the realization of completely decentralized LBSs, being a cross-platform implementation of the ADGT georeferenced P2P overlay scheme. Adgt.js has been accurately implemented adopting standard and open technologies, it has been released online with a free and open source software license and can be used as a web application framework without restrictions.

As part of the Ph.D. activity, the designed peer-to-peer overlay has been intensively evaluated, both through simulations in various and different scenarios of mobility, and by means of field tests performed with the collaboration of several volunteers.

Regarding the simulations, it has been defined a cost-effective approach to software-in-the-loop simulation of pervasive systems and applications. Starting from this simulation methodology, a Java-based simulation platform, which allows to simulate interconnected devices easily, has been realized. The proposed platform provides a general-purpose simulation engine, which includes specific packages to simulate mobility, networking, and energy consumption models. It allows to define general-purpose devices, which can be characterized by multiple network interfaces and protocols, as well as different network and energy models. Hence, with this solution, it has been possible to deeply evaluate the behavior of the ADGT protocol in different scenarios.

With respect to field testing, it has been developed a location-based service for georeferenced instant messaging using the Adgt.js framework. In addition to experiment the versatility of Adgt.js, it has allowed to extrapolate and measure the bahavior of the ADGT peer-to-peer overlay scheme in a real environment.

Both these activities of evaluation have returned positive and comparable results, confirming the fact that ADGT works correctly and that it can be really adopted to realize peer-to-peer LBSs.

Finally, a novel approach for producing proofs-of-location, *i.e.*, digital certificates that attest someone's presence at a certain geographic location, at some point in time whereby LBSs can validate user locations, has been presented. In particular, the approach relies on the blockchain to design a completely decentralized peer-to-peer architecture that guarantees location trustworthiness and preserves user privacy,

at the same time. In fact, to ensure that LBSs work properly, it is necessary that geographic locations claimed by users are factual to prevent, for example, users from accessing georeferenced information to which they are not allowed. With the objective of achieving a system that, at the same time, provides verification of geographic location of its users and ensures a high level of privacy to them, it has been designed a completely decentralized proof-of-location mechanism for location-based peer-to-peer overlay schemes, such as the ADGT.

The consensus mechanism has been evaluated using the aforementioned simulation methodology, obtaining very promising results.

7.1 Future Work

Several research opportunities emerge from the work presented in this Thesis. This section illustrates the most promising directions.

The first direction concerns the extension of the ADGT peer-to-peer overlay scheme, in order to obtain a network topology even more adaptive with respect to peers' mobility. For example, we could consider to modify the neighborhood on the basis of the peer typology (a pedestrian, a vehicle, etc.) and on information related to its itinerary (the destination, the points of interest, etc.).

The second direction regards the exploration of other modern ECMAScript technologies, such as Web Workers and Promises, or the new ORTC (Object Real-Time Communications) W3C specification. Moreover, the performance of the implemented framework, with respect to technological aspects such as battery drain of mobile devices, also compared with previous results obtained in simulation could be investigated.

Finally, the third direction has the purpose to strengthen and deepen the proof-of-location algorithm based on blockchain technology. We plan to investigate other approaches for block mining, also taking into account a mixed Proof-of-Work and Proof-of-Stake solution. Furthermore, it would be interesting to integrate the proof-of-location mechanism into the Adgt.js framework.

Bibliography

[1] Ilya Grigorik. *High Performance Browser Networking*. O'Reilly Media, 2013.

[2] Mark Deakin and Husam Al Waer. From intelligent to smart cities. *Intelligent Buildings International*, 3(3):140–152, 2011.

[3] ISO 19132:2007. Geographic information — location-based services — reference model.

[4] Martin Florian, Felix Pieper, and Ingmar Baumgart. Establishing location-privacy in decentralized long-distance geocast services. *Ad Hoc Networks*, 37, Part 1:110 – 121, 2016. Special Issue on Advances in Vehicular Networks.

[5] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *Commun. Surveys Tuts.*, 7(2):72–93, April 2005.

[6] James Ball. Google maps: thanks for the app, here's my personal data. *The Guardian*, 2012.

[7] Ryan Singel. White hat uses foursquare privacy hole to capture 875k check-ins. *Wired*, 2010.

[8] Andrew Dowell. Tracking women: Now there's not an app for that. *The Wall Steet Journal*, 2012.

[9] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International*

Workshop on Peer-to-Peer Systems, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag.

[10] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.

[11] Filipe Araujo and Luis Rodrigues. Geopeer: a location-aware peer-to-peer system. In *Network Computing and Applications, 2004. (NCA 2004). Proceedings. Third IEEE International Symposium on*, pages 39–46, Aug 2004.

[12] Aleksandra Kovacevic, Nicolas Liebau, and Ralf Steinmetz. Globase.kom - a p2p overlay for fully retrievable location-based search. In *Seventh IEEE International Conference on Peer-to-Peer Computing (P2P 2007)*, pages 87–96, Sept 2007.

[13] Marco Picone, Michele Amoretti, and Francesco Zanichelli. Proactive neighbor localization based on distributed geographic table. In *Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia*, MoMM '10, pages 305–312, New York, NY, USA, 2010. ACM.

[14] Christian Gross, Dominik Stingl, Bjorn Richerzhagen, Andreas Hemel, Ralf Steinmetz, and David Hausheer. Geodemlia: A robust peer-to-peer overlay supporting location-based search. In *2012 IEEE 12th International Conference on Peer-to-Peer Computing (P2P)*, pages 25–36, Sept 2012.

[15] Bernard Heep, Martin Florian, Johann Volz, and Ingmar Baumgart. Overdrive an overlay-based geocast service for smart traffic applications. In *Wireless On-demand Network Systems and Services (WONS), 2013 10th Annual Conference on*, pages 1–8, March 2013.

[16] Giacomo Brambilla, Marco Picone, Michele Amoretti, and Francesco Zanichelli. An adaptive peer-to-peer overlay scheme for location-based services. In *Network Computing and Applications (NCA), 2014 IEEE 13th International Symposium on*, pages 181–188, Aug 2014.

[17] Bernard Heep. R/kademlia: Recursive and topology-aware overlay routing. In *Telecommunication Networks and Applications Conference (ATNAC), 2010 Australasian*, pages 102–107, Oct 2010.

[18] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.

[19] Jonathan Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245, RFC Editor, April 2010.

[20] Jonathan Rosenberg, Rohan Mahy, Philip Matthews, and Dan Wing. Session Traversal Utilities for NAT (STUN). RFC 5389, RFC Editor, October 2008.

[21] Rohan Mahy, Philip Matthews, and Jonathan Rosenberg. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 5766, RFC Editor, April 2010.

[22] Giacomo Brambilla, Michele Amoretti, and Francesco Zanichelli. Adgt.js: a web application framework for peer-to-peer location-based services. In *4th Workshop on Large Scale Distributed Virtual Environments in conjunction of Euro-Par 2016*, LSDVE 2016, Aug 2016.

[23] Justin Uberti, Cullen Jennings, and Eric Rescorla. Javascript Session Establishment Protocol. Internet-Draft, Internet Engineering Task Force, September 2016.

[24] Michele Amoretti and Francesco Zanichelli. P2p-pl: A pattern language to design efficient and robust peer-to-peer systems. *arXiv*, abs/1606.07876, 2016.

[25] Gartner Inc. Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020. `https://www.gartner.com/newsroom/id/2636073`, December 2013.

[26] Michele Amoretti, Marco Picone, and Francesco Zanichelli. Global ambient intelligence: An autonomic approach. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 842–847, March 2012.

[27] Stephen Edward Deering and Robert M. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, RFC Editor, December 1998.

[28] Gabriel Montenegro, Nandakishore Kushalnagar, Jonathan W. Hui, and David E. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, RFC Editor, Semptember 2007.

[29] Zach Shelby, Angelo P. Castellani, and Carsten Bormann. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, 16(undefined):62–67, 2012.

[30] Fredrik Osterlind, Adam Dunkels, Thiemo Voigt, Joakim Eriksson, and Niclas Finne. Cross-level sensor network simulation with cooja. *2006 31st IEEE Conference on Local Computer Networks*, 00(undefined):641–648, 2006.

[31] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 126–137, New York, NY, USA, 2003. ACM.

[32] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455–462, Nov 2004.

[33] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 60:1–60:10, ICST, Brussels, Belgium, Belgium,

2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[34] Elias Weingärtner, Matteo Ceriotti, and Klaus Wehrle. How to simulate the Internet of Things? In *Proceedings of the 11th GI/ITG KuVS Fachgespraech Sensornetze (FGSN 2012)*, Sep 2012.

[35] Michael Kirsche. Simulating the Internet of Things in a Hybrid Way. In *Proceedings of the Networked Systems (NetSys) 2013 PhD Forum*, March 2013. Poster Abstract.

[36] Vilen Looga, Zhonghong Ou, Yang Deng, and Antti YlÃd'-JÃd'Ãd'ski. Mammoth: A massive-scale emulation platform for internet of things. In *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, volume 03, pages 1235–1239, Oct 2012.

[37] Manuel Roman, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy Campbell, and Klara Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, Oct 2002.

[38] Robert Grimm. One.world: Experiences with a pervasive computing architecture. *IEEE Pervasive Computing*, 3(undefined):22–30, 2004.

[39] Anand Ranganathan, Shiva Chetan, Jalal Al-Muhtadi, Roy Campbell, and M. Dennis Mickunas. Olympus: A high-level programming model for pervasive computing environments. In *Third IEEE International Conference on Pervasive Computing and Communications*, pages 7–16, March 2005.

[40] Wilfried Jouve, Julien Bruneau, and Charles Consel. Diasim: A parameterized simulator for pervasive computing applications. In *Mobile and Ubiquitous Systems: Networking Services, MobiQuitous, 2009. MobiQuitous '09. 6th Annual International*, pages 1–2, July 2009.

[41] Abdelsalam Helal, Andreas Mendez-Vazquez, and Shantonu Hossain. Specification and synthesis of sensory datasets in pervasive spaces. In *Computers and*

Communications, 2009. ISCC 2009. IEEE Symposium on, pages 920–925, July 2009.

[42] Giacomo Brambilla, Alessandro Grazioli, Marco Picone, Francesco Zanichelli, and Michele Amoretti. A cost-effective approach to software-in-the-loop simulation of pervasive systems and applications. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pages 207–210, March 2014.

[43] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, June 1999.

[44] Urs Hunkeler, Hong Truong Truong, and Andy Stanford-Clark. Mqtt-s: A publish/subscribe protocol for wireless sensor networks. In *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on*, pages 791–798, Jan 2008.

[45] Simone Cirani, Marco Picone, and Luca Veltri. CoSIP: A Constrained Session Initiation Protocol for the Internet of Things. In Carlos Canal and Massimo Villari, editors, *Advances in Service-Oriented and Cloud Computing*, volume 393 of *Communications in Computer and Information Science*, pages 13–24. Springer Berlin Heidelberg, 2013.

[46] Michele Amoretti, Matteo Agosti, and Francesco Zanichelli. Deus: A discrete event universal simulator. In *Proceedings of the 2Nd International Conference on Simulation Tools and Techniques*, Simutools '09, pages 58:1–58:9, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[47] Michele Amoretti, Marco Picone, Stefano Bonelli, and Francesco Zanichelli. Parallel and distributed simulation with deus. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 600–606, July 2011.

[48] Giacomo Brambilla, Marco Picone, Simone Cirani, Michele Amoretti, and Francesco Zanichelli. A simulation platform for large-scale internet of things scenarios in urban environments. In *Proceedings of the First International Conference on IoT in Urban Space*, URB-IOT '14, pages 50–55, ICST, Brussels, Belgium, Belgium, 2014. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[49] Ivan Seskar, Stetislav Maric, Jack Holtzman, and Jack Wasserman. Rate of location area updates in cellular systems. In *Vehicular Technology Conference, 1992, IEEE 42nd*, pages 694–697 vol.2, May 1992.

[50] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[51] Michele Amoretti, Marco Picone, Francesco Zanichelli, and Gianluigi Ferrari. Simulating mobile and distributed systems with deus and ns-3. In *High Performance Computing and Simulation (HPCS), 2013 International Conference on*, pages 107–114, July 2013.

[52] Alessandro Grazioli, Marco Picone, Francesco Zanichelli, and Michele Amoretti. Collaborative mobile application and advanced services for smart parking. In *2013 IEEE 14th International Conference on Mobile Data Management*, volume 2, pages 39–44, June 2013.

[53] Steve Meloan and Terrence Barr. Java ME 8 and the Internet of Things. `http://www.oracle.com/technetwork/articles/java/ma14-java-me-embedded-2177659.html`, April 2014.

[54] Giacomo Brambilla, Michele Amoretti, and Francesco Zanichelli. Using block chain for peer-to-peer proof-of-location. *arXiv*, abs/1607.00174, 2016.

[55] Vincent Lenders, Emmanouil Koukoumidis, Pei Zhang, and Margaret Martonosi. Location-based trust for mobile user-generated content: Applications, challenges and implementations. In *Proceedings of the 9th Workshop*

on Mobile Computing Systems and Applications, HotMobile '08, pages 60–64, New York, NY, USA, 2008. ACM.

[56] Stefan Saroiu and Alec Wolman. Enabling new mobile applications with location proofs. In *Proceedings of the 10th Workshop on Mobile Computing Systems and Applications*, HotMobile '09, pages 3:1–3:6, New York, NY, USA, 2009. ACM.

[57] Justin Manweiler, Ryan Scudellari, and Landon P. Cox. Smile: Encounter-based trust for mobile social services. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 246–255, New York, NY, USA, 2009. ACM. URL: `http://doi.acm.org/10.1145/1653662.1653692`, `doi:10.1145/1653662.1653692`.

[58] Zhichao Zhu and Guohong Cao. Toward privacy preserving and collusion resistance in a location proof updating system. *IEEE Transactions on Mobile Computing*, 12(1):51–64, January 2013.

[59] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. `https://bitcoin.org/bitcoin.pdf`.

[60] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. 2012. `https://peercoin.net/assets/paper/peercoin-paper.pdf`.

[61] Rasib Khan, Shams Zawoad, Md Munirul Haque, and Ragib Hasan. *'Who, When, and Where?' Location Proof Assertion for Mobile Devices*, pages 146–162. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

Acknowledgements

I want to thank first my supervisors, Prof. Francesco Zanichelli and Prof. Michele Amoretti, for supporting me with their experience, providing suggestions, and interesting ideas. Especially, thank you for your infinite patience, for understanding and for your loyalty.

Of course, a huge thanks goes to my whole family, but most of all, to my brother for having been by my side every time I needed help. A big thank goes to Chiara, for loving me, taking care of me, and supporting me all the time.

Finally, I want to thank Alessandro Grazioli, a great friend known during this Ph.D., without whom this thesis would not have been possible. Thank you for all the wonderful time we spent together.

Thank you all,
Giacomo