POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA
DOCTORAL PROGRAMME IN INFORMATION TECHNOLOGY

A MIDDLEWARE FOR PEER-TO-PEER INTERACTION OF SMART THINGS

Doctoral Dissertation of:
Naser Derakhshan

Supervisor:
Prof. Luciano Baresi

Tutor:
Prof. Donatella Sciuto

The Chair of the Doctoral Program:
Prof. Andrea Bonarini

2016 – XXIX

This work was supported by the Joint Open Lab S-Cube TIM/Telecom Italia S.p.A. — Innovation division, Milano, Italy

Acknowledgment

First, I would like to express my sincere gratitude to my advisor Prof. Luciano Baresi for his patience, motivation, and immense knowledge that make my Ph.D. experience productive and stimulating. I have learned a lot during our discussions, and that has contributed significantly to the development of this work. Besides my advisor, I would like to thank Prof. Sam Guinea and Dr. Valerio Panzica La Manna who have helped me initially to formulate the basis for this research and then worked together with me in pursuing it.

I am grateful to Telecom Italia for funding my Ph.D. studies under the initiative of Joint Open Lab (S-CUBE). This work would not have been possible without their financial and technological support. Specifically, I would like to thank Massimo Valla, Francesco Arenella and Adnan Shahzada for the useful and productive collaboration. Moreover, I would like to thank Gabriella Giardina who provided a laboratory environment at Samsung District in Milan for test and evaluation of my middleware.

My sincere thanks also goes to Prof. Paolo Bellavista and Prof. Henry Muccini for accepting to review my manuscript and giving their valuable comments. I also want to offer my gratitude to my tutor Prof. Donatella Sciuto.

To my wonderful parents and darling wife

Abstract

In recent years we have been assisted by the rapid development of small, low-power, and low-cost wireless computation/communication devices, which have served as enablers for the so-called *Internet of Things* (IoT). Within the Internet of Things, devices adopt wireless communication to cooperate to provide services and added value to users, probably in large-scale dynamic environments. This type of device-centric cooperation can be interpreted as social interaction between enabled devices instead of between people. Smartphones and tablets are clearly the most widely-known examples of such devices; however, there are also many others (e.g., home and office appliances or wearable devices) and much more will be available shortly.

Devices that can connect to the Internet can exploit cloud-based services to enable interaction in an IoT scenario. However, an Internet connection might not always be available, or it might be too expensive to use. It is clear that within the Internet of Things we cannot impose the requirement of having an always-on functional Internet connection. Instead, we need to shift our focus to a new wave of interaction called *proximity-based interactions*. Applications can benefit from this type of interaction when users and devices are physically close to one another. They do not rely on an Internet connection; instead, they operate in an infrastructure-less scenario, possibly interacting in a peer-to-peer (P2P) manner. To enable proximity-based interactions, devices should be equipped with one or more P2P communication protocols such as Bluetooth, BLE, NFC, Wi-Fi Direct, 6LowPan, ZigBee, and ultrasonic.

Among P2P communication protocols for mobile devices, Wi-Fi Direct has recently gained attention because it is widely available on smart devices, specifically smartphones, and also does not have the critical problems of traditional ad-hoc Wi-Fi such as speed, security, stability, scalability, and power consumption. Although many works have already tried to exploit Wi-Fi Direct in social interaction between proximal smart devices, there is not any work that attempts to operate Wi-Fi Direct in large-scale dynamic application domains.

This thesis bridges this gap by proposing a middleware infrastructure, called **MAGNET**, that aims to provide reliable and stable P2P connectivity between large numbers of smart devices in a dynamic environment without user intervention. The goal is to

remove the known limitations and offer a better means to exploit this technology. This goal is achieved by managing devices in several Wi-Fi Direct groups and ensuring that each device is part of a Wi-Fi Direct group in any circumstance (i.e. saturation, moving, and failing). In the second step, MAGNET oversees inter-connecting Wi-Fi Direct groups and provides multi-hop connectivity between a large number of devices. To evaluate the effectiveness of MAGNET, a Wi-Fi Direct simulator has also been developed, called **WiDiSi**. The proposed solution has been tested on real Android devices and the simulation environment. Realistic scenarios are utilized in the thesis to showcase and evaluate the key features of the proposed solution. The evaluation results illustrate the effectiveness of such a solution in providing a stable communication between a large number of mobile devices using Wi-Fi Direct.

Contents

1 Introduction **5**
- 1.1 Internet of Things . . . 5
- 1.2 Problem and Research Questions . . . 8
- 1.3 Research Objectives . . . 10
- 1.4 Major Contributions . . . 10
- 1.5 Thesis Structure . . . 11

2 Proximity-Based Software Technologies **13**
- 2.1 Definition . . . 14
- 2.2 Communication Protocols . . . 15
  - 2.2.1 Wi-Fi Direct . . . 15
  - 2.2.2 Bluetooth Low Energy . . . 19
  - 2.2.3 LTE-Direct . . . 21
  - 2.2.4 Thread and 6LowPAN . . . 22
- 2.3 Middleware Technologies . . . 24
  - 2.3.1 AllJoyn . . . 24
  - 2.3.2 IoTivity . . . 26
  - 2.3.3 Google Nearby . . . 28
  - 2.3.4 Prototypal Solutions . . . 30
- 2.4 Discussion and Comparison . . . 31

3 Wi-Fi Direct Simulation Environment **37**
- 3.1 Problem and Motivation . . . 37
- 3.2 PeerSim . . . 39
- 3.3 Proposed Simulator Architecture . . . 39
- 3.4 Implementation Details . . . 40
  - 3.4.1 Wi-Fi Direct Interface . . . 41
  - 3.4.2 Network Dynamism and Proximity Manager . . . 44
  - 3.4.3 Node Initializers . . . 44
  - 3.4.4 IEEE 802.11 . . . 45
  - 3.4.5 Network Visualization and Logging . . . 45

3.5 Evaluation 49

4 Proposed Middleware Infrastructure **53**
4.1 Architecture 53
4.1.1 Device Abstraction 55
4.1.2 Group Abstraction 55
4.1.3 Communication Abstraction 56
4.2 Implementation 57

5 Evaluation **63**
5.1 Qualitative Evaluation 64
5.1.1 Case Study One: A Ceremony 64
5.1.2 Case Study Two: A Shopping Mall 64
5.2 Quantitative Evaluation 69
5.2.1 Case Study Three: Eight Physical Devices (SCUBE Lab) 70
5.2.2 Case Study Four: Eighteen Physical Devices (Samsung Lab) . . 73
5.2.3 Case Study Five: Simulations on WiDiSi 77
5.3 Threats to validity 80

6 Conclusion and Future Directions **81**
6.1 Answers to Research Questions 81
6.1.1 Research Question One [Q.1] 81
6.1.2 Research Question Two [Q.2] 82
6.1.3 Research Question Three [Q.3] 83
6.1.4 Research Question Four [Q.4] 83
6.2 Future Directions 83

Bibliography **85**

List of Figures

1.1 IoT — Explosion of Connected Possibility [1] 6
1.2 IoT — in every aspects of our lives. Picture is taken from [1] 6
1.3 Internet of things — 6A connectivity 7
1.4 Connectivity in isolated areas (a), emergency situations (b), high density (c), or when a special quality of service is needed (d). — Pictures (a) and (b) are taken from [15] and pictures (c) and (d) are taken from Tasnim News Agency and CeBIT respectively 8

2.1 Example Device and Service Discovery procedures for a P2P Device . . 16
2.2 P2P Concurrent device 18
2.3 GATT data hierarchy [82] 20
2.4 LTE and LTE Direct network topology 22
2.5 LTE D2D Scenarios [18] 22
2.6 Basic Thread Communication Stack [17] 23
2.7 Basic Thread Network topology [17] 23
2.8 AllJoyn Bus Architecture 25
2.9 IoTivity Architecture 27
2.10 SSM Context Diagram [54] 28
2.11 Nearby Message API overview 29

3.1 Overview of WiDiSi 41
3.2 Main components of the WiDiSi implementation 42
3.3 Roles and status of the nodes 46
3.4 A screen shot from the main visualization window 47
3.5 Real time control, monitor and logging windows 48
3.6 Time needed for the standard deviation to become less than 0.15 (Scenario 1) 51
3.7 Time needed for a 10-minute simulation with different numbers of nodes (Scenario 2). We assume, a second corresponds to 10 PeerSim cycles . 51
3.8 Maximum number of possible simultaneously connected devices for various radio ranges 52

4.1 MAGNET architecture 54
4.2 Example group topology and bridges 58
4.3 Our routing algorithm for MAGNET 61
4.4 Inter-Group communications with different strategies - (a) Group owner becomes a legacy client in another group and (b) A P2P client becomes a legacy client in another group (our solution) 62

5.1 A funeral in Iran 65
5.2 A funeral in Iran — Simulated using WiDiSi 66
5.3 A funeral in Iran — The network status 67
5.4 Case Study 2: Shopping mall 67
5.5 MAGNET reaction to Event.1 and Event.4 68
5.6 MAGNET reaction to Event.2 69
5.7 MAGNET reaction to Event.3 70
5.8 MAGNET reaction to Event.5 71
5.9 Average delay for rehabilitating the network to working condition after various events (8 devices) 72
5.10 The relationship between delays in message delivery and the number of hops (8 devices) 72
5.11 The topology of the network for measuring the baseline for message drop rate 73
5.12 Experiment on eighteen devices at Samsung Lab 74
5.13 Average delay for rehabilitating the network to working condition after various events (18 devices) 75
5.14 The relationship between delays in message delivery and the number of hops (18 devices) 76
5.15 The maximum and the minimum message delivery delays per hop (18 devices) 76
5.16 The standard deviation of delay values per hop (18 devices) 77
5.17 A section of the network with four separate clusters 78
5.18 Relationship between devices' mobility and network connectivity . . . 79

List of Tables

2.1 A comparison in terms of mobility and large-scale support 31
2.2 Comparison in a nutshell 35

5.1 Qualitative Evaluation 63
5.2 Quantitative Evaluation 63

CHAPTER 1

Introduction

In this chapter, we briefly define and elaborate on the concept of the Internet of Things and introduce the scope of the work.

1.1 Internet of Things

In recent years we have assisted to the rapid development of small, low-power, and low-cost wireless computation/communication devices, which have served as enablers for the so-called *Internet of Things* (IoT). Within the Internet of Things, devices adopt wireless communication to cooperate to provide services and added value to users. Smartphones and tablets are clearly the most widely-known examples of such devices; however, there are also many others (e.g., home and office appliances or wearable devices) and many more will be available in the near future. Erikson [5] foresees, that there will be more than 50 billion connected devices in 2020 that are interacting with each other and will truly enable the Internet of Things. Fig. 1.1 illustrates such trends.

The Internet of Things promises to enable new and unforeseen scenarios in many different domains. It will allow us to enhance logistics and public transportation, to create smart exhibition centers and museums, to develop more efficient and effective hospitals and offices, and to truly realize home automation and the promise of connected cars. Fig. 1.2 illustrates the IoT in every aspect of our lives.

Researchers have described the concept of the IoT from various perspectives. For instance, the Internet Research Task Force (IETF) [53] believes that the IoT will connect objects around us "*to provide seamless communication and contextual services provided by them. Development of RFID tags, sensors, actuators, mobile phones make it possible to materialize IoT which interact and co-operate with each other to make the service better and accessible anytime, from anywhere.*" A more detailed definition

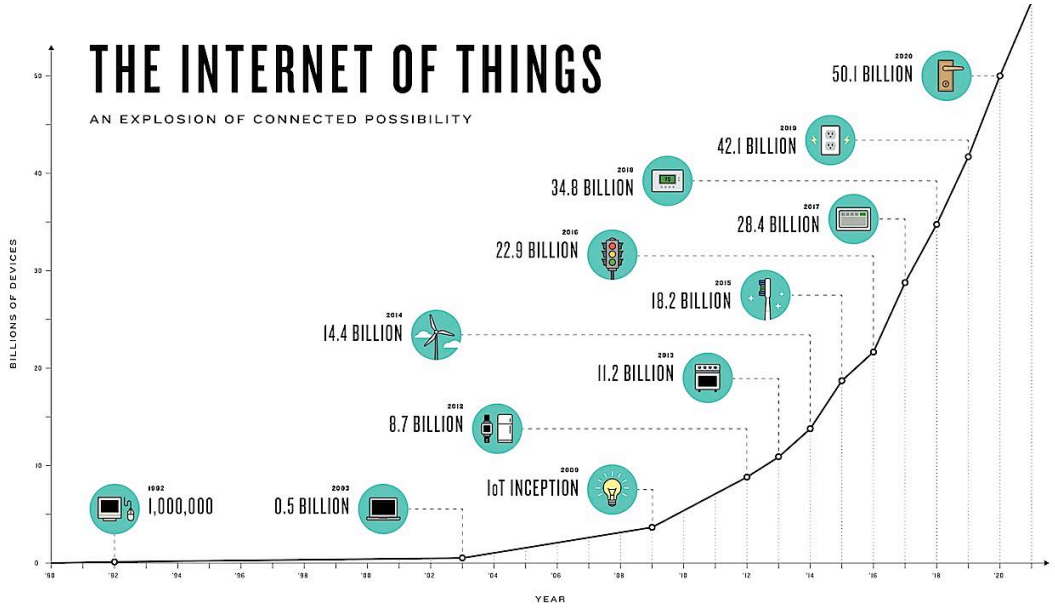


Figure 1.1: *IoT — Explosion of Connected Possibility [1]*

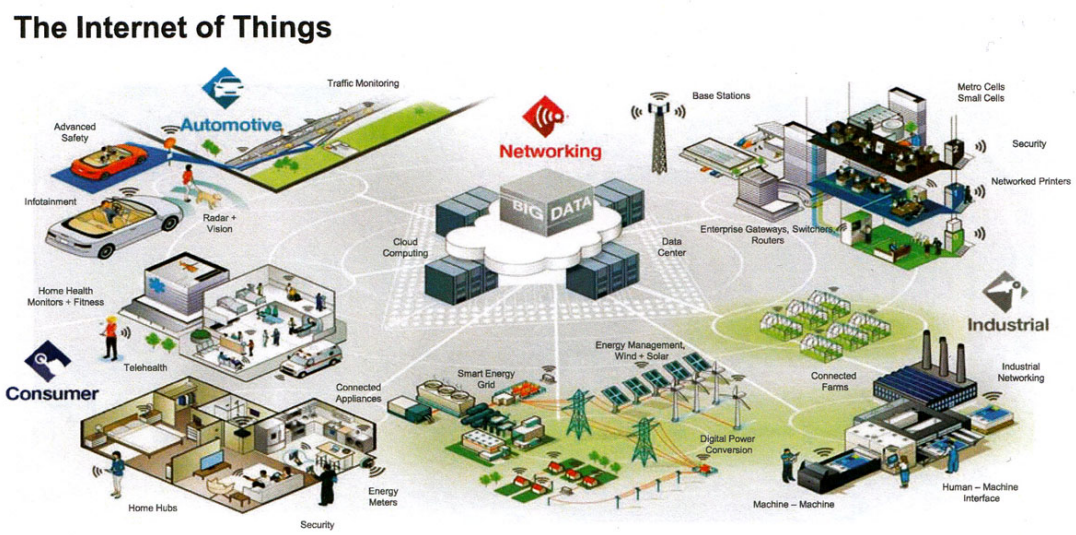


Figure 1.2: *IoT — in every aspects of our lives. Picture is taken from [1]*

has been stated by Vermesan et al. [84]. They defined the IoT as "*a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual 'things' have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network. In the IoT, 'things' are expected to become active participants in business, information and social processes where they are enabled to interact and communicate among themselves and with the environment by exchanging data and information 'sensed' about the environment, while reacting autonomously to the 'real/physical world' events and influencing it by running processes that trigger actions and create services with or without direct human intervention.*" Based on this definition, the Internet of Things could allow people and things to be connected anytime, anyplace, with anything and anyone, ideally using any path/network and any service(Fig. 1.3). This definition is also stated in the ITU vision of the IoT, according to which: "From anytime, anyplace connectivity for anyone, we will now have connectivity for anything" [84].

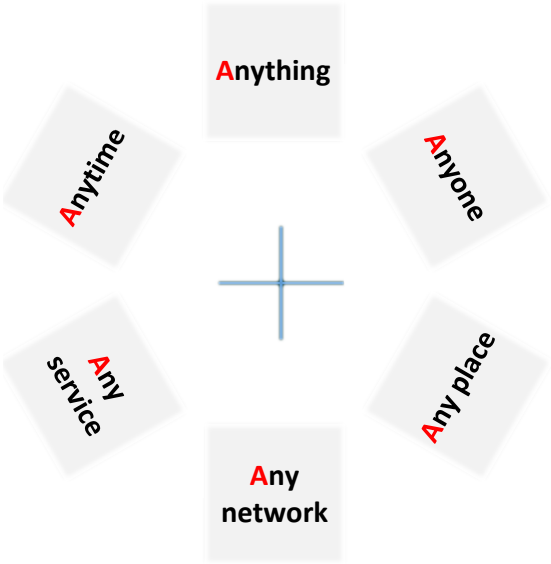


Figure 1.3: *Internet of things — 6A connectivity*

From the software perspective, regardless of the hardware exploited for connectivity and communication, the interaction can be either based on a fixed infrastructure (possibly interacting via the Internet) or an infrastructure-less peer to peer(P2P) manner. Devices that can connect to the Internet can exploit cloud-based services to enable interaction in an IoT scenario. Such services can be of many different types: from communication services to services providing remote control of physical objects, from environmental data collection to analysis services, from personal fitness services to the national health-care system, and many others. Although cloud-based interaction can support a broad range of applications, the *logical* centralization and the intrinsic need for Internet connectivity can hamper the realization of some significant IoT (Internet of Things [56, 78]) scenarios. For example, so-called *proximity-based* applications exploit a P2P communication paradigm and often do not require Internet connectiv-

ity. The nodes (devices) that are part of the system only communicate with the other nodes in their physical proximity and can possibly exploit multi-hop routing to enable infrastructure-less communication between a large number of nodes.

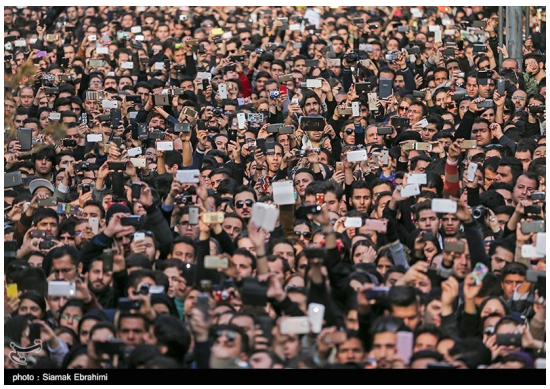
In this thesis, we concentrate on the infrastructure-less communication and the properties and the challenges of such interaction between devices.

1.2 Problem and Research Questions

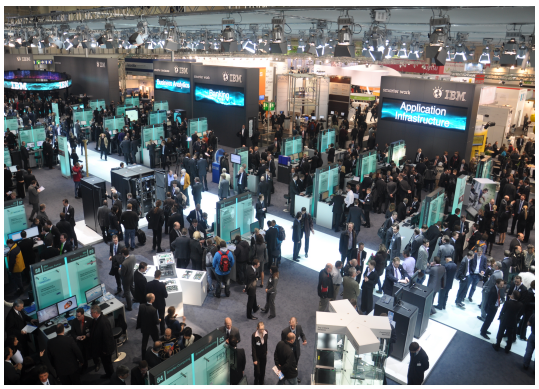
Many software systems for the interaction of people and things exploit the "usual" Internet protocols and some centralized cloud-based services. However, an Internet connection might not always be available, or it might be too expensive to use. This can happen in many cases. For example, devices might need to operate where there is no Internet connectivity (e.g., in isolated areas, on boats and airplanes, Fig. 1.4a), where Internet access has been compromised (e.g., a disaster recovery scenario, Fig. 1.4b), or where device density is compromising the use of the network (Fig. 1.4c). Moreover, slow connectivity might also hamper device interaction (Fig 1.4d).

(a) A group of tourist in a desert may connect via a P2P communication technology

(b) An earthquake in Nepal — first responders may search for SOS signals



(c) Cellular network had been saturated when a large number of users wanted to share images of a famous singer funeral in Iran on November, 2014.



(d) An interaction between smartphone and large screens in a exibition may need higher speed than the one provided by the infrastructure

Figure 1.4: *Connectivity in isolated areas (a), emergency situations (b), high density (c), or when a special quality of service is needed (d). — Pictures (a) and (b) are taken from [15] and pictures (c) and (d) are taken from Tasnim News Agency and CeBIT respectively*

Both industry and academia have presented different solutions to support infrastructureless *social* interactions. They are either user applications such as FireChat[1], which is a proximity-based messaging application, or middleware technologies such as AllJoyn [20], IoTivity [44], or Google Nearby [39]. In addition to these industry-supplied solutions, many other prototypal solutions enable the P2P social interactions between proximal devices (e.g., [29, 30, 42, 60, 75, 90]) and exploit Wi-Fi ad-hoc, Wi-Fi Direct, and Bluetooth. In chapter 2, a short survey discuss all these technologies as well as the most promising wireless P2P communication protocols that can enable proximity-based social interaction between smart devices.

Among P2P communication protocols for mobile devices, Wi-Fi Direct [2] was introduced by the Wi-Fi Alliance in 2010 and, unlike Wi-Fi, does not depend on a fixed access point (AP). It has longer range and higher speed than Bluetooth and also much better power management [66] and security and connection reliability than traditional Wi-Fi ad-hoc (iBSS) mode [86]. Moreover, unlike Wi-Fi iBSS, Wi-Fi Direct does not require specific hardware requirements for operation [83, 88]. These characteristics make Wi-Fi Direct a suitable candidate for many application domains such as mobile social networking [87], file sharing [24], LTE offloading [23, 68–70], disaster recovery [67], and inter-vehicular communications [41].

Although Wi-Fi Direct offers some powerful features for the discovery of and communication between nearby devices, it has some critical limitations that prevent the use of this technology in large-scale dynamic scenarios. The mobility of smart devices, along with devices that disappear because they run out of battery, can make Wi-Fi Direct connectivity unstable. In addition to this, there is a limit to the number of devices that can be part of a Wi-Fi Direct group[2], that is, that can be connected. This means that as soon as a group becomes saturated, it cannot accept any additional member, which would then be rejected. All these problems require human intervention to re-establish broken connections or manage saturated groups.

Below we have summarized the current challenges of the exploitation of Wi-Fi Direct in social interaction of smart devices, and the open research questions that we address in this thesis:

Q.1 Is Wi-Fi Direct a suitable candidate to enable the proximity-aware interaction of smart things in an IoT scenario? (Chapter 2)

Q.2 How is it possible to remove Wi-Fi Direct intrinsic limitations and exploit it to connect a large number of devices? (Chapter 4)

Q.3 How is it possible to manage mobility in a Wi-Fi Direct network in order to offer a stable connection to the application? (Chapter 4)

Q.4 How is it possible to test and evaluate the effectiveness of proposed solutions in realistic large-scale dynamic situations? (Chapter 3 and Chapter 5)

To answer the first research question[Q.1], we performed a survey of the most promising technologies that can support proximity-based applications, and discuss their current strengths and weaknesses. This overview in Chapter 2 is a first step in helping

[1] `https://firech.at`

[2] Although the Wi-Fi P2P specification does not set any limit on the number of nodes in a group, recent versions of Android APIs limit this number to four to maintain reasonable performance.

developers choose the most appropriate technology stack for their applications, depending both on the applications' scenarios and on their functional and non-functional requirements. This survey leads us to a proper communication technology for the social interaction of large-scale smart mobile devices. The second research question [Q.2] emphasizes the limitations of Wi-Fi Direct protocol when facing a large number of devices. Since a Wi-Fi Direct group can host only a few clients, a solution is needed to manage connections when the number of devices passes the maximum number. The third question [Q.3]. is related to situations where mobility of devices or device failures may cause a whole group termination. The last question is about a suitable simulation framework for realistic tests on a large number of devices with various mobility patterns.

In this thesis, we answer all these questions by introducing a middleware infrastructure and a comprehensive simulation environment for the evaluation.

1.3 Research Objectives

The main research goal of this thesis can be summarized as follow:

A self-organizing distributed middleware infrastructure that aims to provide reliable and stable P2P connectivity between large numbers of smart devices in a dynamic environment without user intervention. The goal is to remove the known limitations and offer a better means to exploit Wi-Fi Direct technology. Moreover, a simulation environment is also required to evaluate the proposed solutions

The intrinsic limitations of Wi-Fi Direct specification prevent the use of this technology in large-scale dynamic application domains. One the one hand, the mobility of devices, along with devices that disappear because they run out of battery or because of unknown internal errors, will interrupt the connections. If a device gets out of the radio range of the group, it will get disconnected. The situation is worse when the group owner leaves the proximity of the group. In this case, the whole Wi-Fi Direct group will be terminated. Moreover, when several unconnected devices come into proximity with each other, a group owner election should be performed to choose one device as a group owner and the others as clients. All devices should also be aware of unsuccessful connection requests. All these events require user intervention to create a connection or reestablish a broken connection. On the other hand, other limitations are related to situations where there are a large number of devices that want to form a solid P2P network. The Wi-Fi Direct specification does not introduce any procedure for connecting a large number of devices. To name a few, group saturation, multi-group connections, and multi-hop communication are examples of such limitations. To the best of our knowledge, there is not solution that addresses all these limitations and offers a concrete solution for the application developer to exploit Wi-Fi Direct in large-scale dynamic application domains.

1.4 Major Contributions

The major contributions of this thesis can be summarized as follows:

- ✓ A comprehensive survey on all the key proximity-aware software technologies. This overview is a first step in helping developers choose the most appropriate technology stack for their applications, depending both on the applications' scenarios and on their functional and non-functional requirements
- ✓ A self-organizing middleware infrastructure, called **MAGNET**, that provides reliable and stable P2P connectivity between large numbers of smart devices in a dynamic environment. This middleware abstracts the multi-hop communication process by autonomously maintaining connectivity between devices. *MAGNET* also provides a discovery mechanism that exploits the MAC address of the different devices or the services they offer.
- ✓ A research-oriented prototype simulator for Wi-Fi Direct networks, called **WiDiSi**, that allows Android Wi-Fi Direct applications to be easily tested in large-scale dynamic scenarios.
- ✓ Suitable abstractions, APIs, and documentation that help application developers to integrate MAGNET easily and test their Android applications in the simulation environment.

1.5 Thesis Structure

The thesis is constituted of six chapters. In this section, we give an overview of these chapters.

Chapter 2 surveys the state of the art in the field of proximity-based applications. This is a study on the most promising technologies that can support proximity-based applications, and we discuss their current strengths and weaknesses.

Chapter 3 describes WiDiSi, a research-oriented prototype simulator for Wi-Fi Direct networks. WiDiSi is provided as an extension of PeerSim, a widely-used, open-source simulation framework for large-scale peer to peer networks. WiDiSi's main goal is to allow Android Wi-Fi Direct applications to be easily tested in large-scale dynamic scenarios. In this chapter, we explain how WiDiSi was modeled and present its general architecture. We also provide evaluation results that show the correctness and the capabilities of the simulator.

Chapter 4 presents *MAGNET*, a novel middleware infrastructure that exploits Wi-Fi Direct to provide a reliable and stable communication means for large numbers of mobile devices. It also discusses the implementation details of MAGNET. In particular, this chapter describes the implementation challenges and the way to overcome the Wi-Fi Direct limitations.

Chapter 5 discusses the test scenarios. We tested *MAGNET* on both our simulator for Wi-Fi Direct-based systems and on real Android devices. The evaluation results illustrate the effectiveness of the proposed middleware in discovering and maintaining the connectivity in large-scale dynamic scenarios.

Chapter 6 concludes the thesis by listing the contributions. Furthermore, it discusses the possible future work in this area.

CHAPTER 2

Proximity-Based Software Technologies

It is clear that with the Internet of Things we cannot impose the requirement of having an always-on functional Internet connection. Instead, we need to shift our focus to a new wave of IoT-enabled applications called *proximity-based applications*. Proximity-based applications are applications that can be used when users and devices are physically close to one another. They do not rely on an Internet connection; instead, they operate in an infrastructure-less scenario, possibly interacting in a peer-to-peer (P2P) manner. They also naturally support context-aware applications, i.e., applications in which the devices are aware of their surrounding environment, can gather information from it, and have a lasting impact on it. Examples of such applications already appear in our every-day life. For example, smart television sets support P2P streaming of media content from laptops and mobile devices; while proximity-based messaging applications, such as FireChat[1], allow users to communicate in crowded areas without Internet access.

Unfortunately, these services typically rely on ad-hoc communication solutions and are difficult to combine and integrate into new services. A common, extensible, and interoperable middleware that can support the interaction of devices (services) that are within proximity of one another without using the Internet is needed. Fortunately, some first steps in this direction have already been made, both in terms of standardized communication protocols and middleware infrastructures.

In this chapter, one the one hand, we provide an informed look at the most promising communication protocols and middleware infrastructures, and analyze their current strengths and weaknesses. On the basis of this analysis, we then discuss and compare the different technologies, to help developers choose the ones that are more appropriate according to their application scenarios, and to their functional and non-functional

[1] `https://firech.at`

requirements. On the other hand, we also introduce the related works in the literature and discuss their limitations in our working environment. At the end of this chapter, we expect that the reader get familiar with the key technologies and protocols that enable the proximity-based interaction and understand the limitations of the exploitation of the current solutions in large scale dynamic application domains.

The rest of this chapter is organized as follows. Section 2.1 provides some detailed examples of proximity-based applications. Section 2.2 describes the current crop of standardized communication protocols that can be used in proximity-based applications. Since the rest of this thesis is based on the Wi-Fi Direct, it has been explained in greater details than the other protocols. In Section 2.3 we describe the most promising middleware infrastructures for proximity-based applications while section 2.4 compares and discusses the different technologies.

2.1 Definition

Proximity-based applications leverage information and functionality from the surrounding space. To do so, they need to be able to *discover* what information and services are provided by the proximal devices. Being able to collect proximity-based information is essential for the creation of truly context-aware applications.

After discovery has been carried out, proximal devices can connect and interact in a seamless way. The objective is to create a viable P2P communication network for the application, one that will be able to adapt to the users' mobility, and cover scenarios where new peers can appear or disappear without notice.

For example, traditional social networking applications use the Web to create and maintain social connections among people, such as friends, family members, and colleagues. These connections persist across space (e.g., the people are physically far away from one another), and across time (e.g., people connect at different times). The introduction of proximity-based communication will open new scenarios for existing social networks and enable the creation of applications that discover and create (temporary) connections with nearby people. For example, people in a party can discover other people with similar interests, or by publicly advertised social profiles. In a business fair or a conference, business connections can be created to foster constructive interactions. Furthermore, these kinds of applications will be able to operate even when The Internet is not available or overloaded; so, for example, they could be used by users that are traveling on a train or at a concert.

Scenarios involving public security and disaster recovery require that selected devices be able to interact promptly when Internet-based connectivity is not available. Proximity-based applications can play a crucial role in this domain. For example, users could notify their presence within a given proximity, and be located faster by first responders. Officials could also use proximity-based applications to exchange public safety information reliably.

More in general, one can highlight the positive impact that the use of proximity-based connections has on security and privacy. Indeed, to attack a proximity-based connection, the attacker needs to be in the proximity of the devices; this is not necessarily a requisite when attacking a device that is connected to the Internet.

The introduction of proximity-based applications also opens new scenarios for ad-

vertising. Users might be notified of the presence, within a store, of someone that matches their preferences or needs. Retail shops and restaurants might decide to provide limited offers and coupons on the basis of the clients currently on their premises. This technology can also help spread detailed information about the products in a supermarket, and let users make a more informed decision on what items to buy.

More generally, proximity-based capabilities can turn passive and static entities into active and dynamic ones that are able to act and react according to the different contexts and situations. This is imperative in many scenarios that address advanced logistics where transported goods can announce themselves and create ad-hoc networks to exchange information about their status and needs. For example, they might exploit these networks to notify anomalous conditions (e.g., a temperature is too high, or tanks storing incompatible substances are positioned too close to one another).

2.2 Communication Protocols

This section provides an overview of the different network layer protocols that can be used to develop proximity-based applications. The selection is based on two main criteria: 1) proximity awareness, and 2) being widely available in smart devices, specially smartphones. Among these protocols, we have chosen Wi-Fi Direct because of its interesting features. These features have been discussed in detail in section 2.2.1.

2.2.1 Wi-Fi Direct

Wi-Fi Direct[2] [2] is a technology that was introduced by the Wi-Fi Alliance in 2010 as a complementary technology to traditional Wi-Fi ad-hoc; it allows Wi-Fi enabled devices to communicate with each other —at regular Wi-Fi speeds and ranges— without having to depend on a fixed access point (AP). Wi-Fi Direct improves Wi-Fi ad-hoc by means of WPA2 (Wireless Protected Access) security, power management protocols, and the use of the full communication range and speed of IEEE 802.11 a/b/g/n. Scalability and stability have also been improved by introducing the role of *Group Owner*, which acts as access point In Wi-Fi Direct. Any enabled device can play the role of AP, and if so is called *group owner*. The group owner provides the communication infrastructure for all the other devices within its range. Nowadays Wi-Fi Direct can be found on all the major OS platforms, although to various degrees of implementation. While Android provides a full Wi-Fi Direct implementation, iOS devices can exploit Apple's Multipeer Connectivity framework [50] to send message-based data, streaming data, and resources (such as files) to other iOS devices.

The operation of a Wi-Fi Direct network is organized around four distinct features: discovery, group formation, communication, and power management.

Device and Service Discovery Wi-Fi Direct follows a two-step asynchronous *device/service discovery* scheme. During the first step –called the searching phase– a device broadcasts a request for the MAC addresses of all the devices that are within its range; all the available 802.11 channels (channels 1, 6, and 11) are used. During the second phase –called the listening phase– it waits for their responses. During this second

[2]Wi-Fi Direct and Wi-Fi P2P are used interchangeably in this thesis. Wi-Fi Direct is the commercial name while Wi-Fi P2P is the technical name used in the specification and in Android developer website

phase, the devices can optionally issue a second request to the discovered peers, to obtain their service information. Figure 2.1 illustrates the sequence diagram of a *Device and Service Discovery* procedure between two P2P devices.

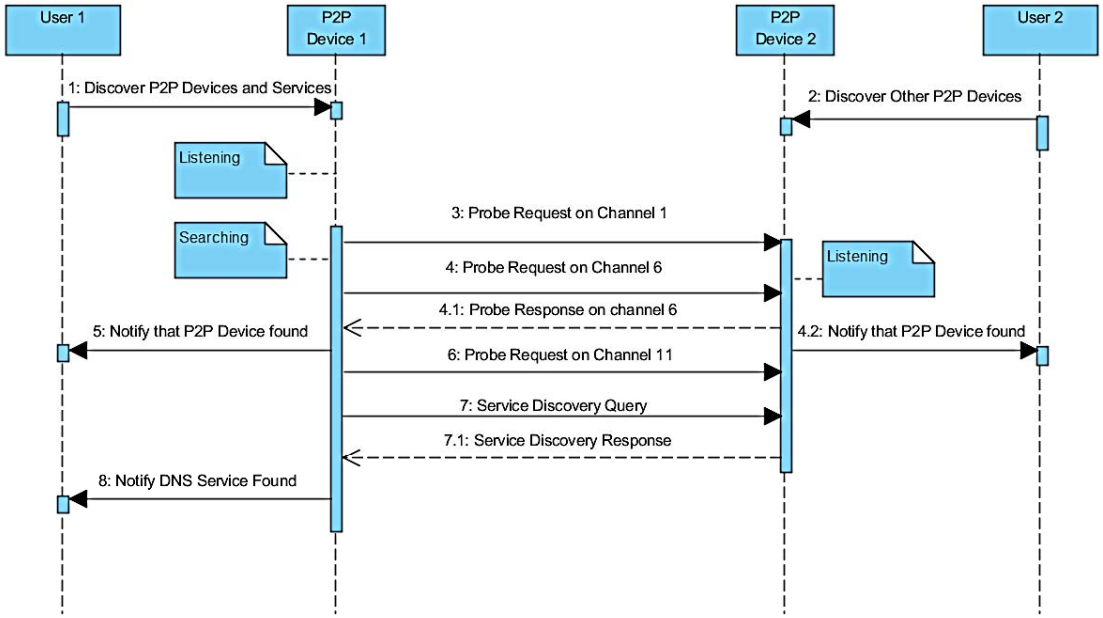


Figure 2.1: *Example Device and Service Discovery procedures for a P2P Device*

Asynchronous discovery means that the devices are not aware of the proximal devices that may be *advertising* their presence. According to this mechanism, two devices can only find one another if, at the time one device is listening on a channel, the other is advertising its presence on that same channel. Devices continuously alternate their searching and listening states, and remain in a given state for a randomly-set amount of time, typically between 100ms and 300ms. This type of advertisement and discovery, in addition to the signal congestion when several devices advertising at simultaneously, cause high uncertainty about the time needed for a device to discover the others in proximity.

Asynchronous discovery is considered to be quite slow; experimental results from Qualcomm show that they were able to discover 369 devices in 82 seconds [51]. This is slow if compared to LTE Direct's discovery, which can discover up to 7200 devices in less than a second. The main difference is that LTE Direct's discovery is synched with the operator's base stations. This means that LTE Direct devices know when they should be listening and when they should be advertising.

On top of device discovery, Wi-Fi Direct also supports *service discovery*, i.e., a means to discover the application-level functionalities that are provided by the proximal devices. Service discovery is based on the *Generic Advertisement Service* protocol (GAS, [25, 72]). Thanks to this protocol devices can exchange information about application-level services at the OSI data-link layer. In practice, this means being able to discover proximal services even before a connection is created with a specific device. In this thesis, we utilized this feature of Wi-Fi Direct advertise and discover device's relevant information before connecting them to each other.

Discovery is periodically repeated to keep the list of proximal devices and services up-to-date, since Wi-Fi Direct does not support automatic notification of devices that leave the proximity or simply fail. In theory, each device should be able to discover any other device/service in its 200-meter radio range. However congestion, signal blockage due to physical obstacles, or noise may decrease the range or the number of discovered elements.

Group formation The topology of a Wi-Fi Direct network is similar to a traditional Wi-Fi network, with the group owner playing the role of the AP. This allows the topology to also support the connection of legacy Wi-Fi devices. In Wi-Fi Direct proximal devices negotiate the role they play: one of them becomes the group owner while the others become clients. This can be achieved in one of the following three ways:

- *Standard* way: this approach exploits device *intention* values, i.e., the devices' willingness to become the group owner. In this approach two devices exchange their intention values, and the one with the highest value becomes the group owner. If they have the same intention value a tie-breaker bit is used. Once the group owner has been identified, the other proximal devices can send connection requests to become a part of the group, or the group owner can send invitations to unconnected devices.
- *Persistent* way: the election of the group owner is similar, but negotiating devices save their credentials so that they can recreate the same topology in the future, without re-performing the negotiation phase.
- *Autonomous* way: a device can decide to act as the group owner without going through negotiation, and then send out invitations to the unconnected devices. This type of group formation is faster than the standard way since it does not involve long scanning and probing activities [32].

A group owner can invite other unconnected Wi-Fi Direct devices to join the group, unconnected devices can send a request to a group owner to join the group, or a connected device can invite an unconnected one to join the group. Moreover, some clients can act as *legacy* ones and try to join a group as if they were joining a "regular" Wi-Fi network: the owner is nothing but the access point they connect to.

The above-mentioned requests may not turn into successful connections due to many reasons: congestion, that is, several requests at the same time, group saturation, or because of power save mechanisms. The application that would like to become part of a group is in charge of checking if the connection has been established or it must send another request.

A single device can also play both the role of a client and a group owner at the same time using radio time sharing, or by means of two separate frequencies if the device has multiple physical radios [33]; however, it has not been discussed in the Wi-Fi direct specification and it is not allowed in Android. Once a device becomes the group owner, the role cannot be transferred to other members. This means that whenever the group owner leaves the group or is disconnected, the entire network becomes corrupt. The remaining clients have to restart a new election process and find a new group owner.

A device can operate concurrently on a Wi-Fi Direct network and a "standard" Wi-Fi one. The concurrent operation requires a device to support multiple MAC addresses.

As an example Figure 2.2 shows a P2P concurrent device that has one MAC address operating as a Wireless LAN stationary and the second MAC address operating as a P2P device. The dual MAC functionality can be provided via two separate physical MAC entities each associated with its own PHY entity, two virtual MAC entities over one PHY entity, or any other approach. Implementation of multiple MAC functionality has not discussed in the Wi-Fi P2P specification [21].

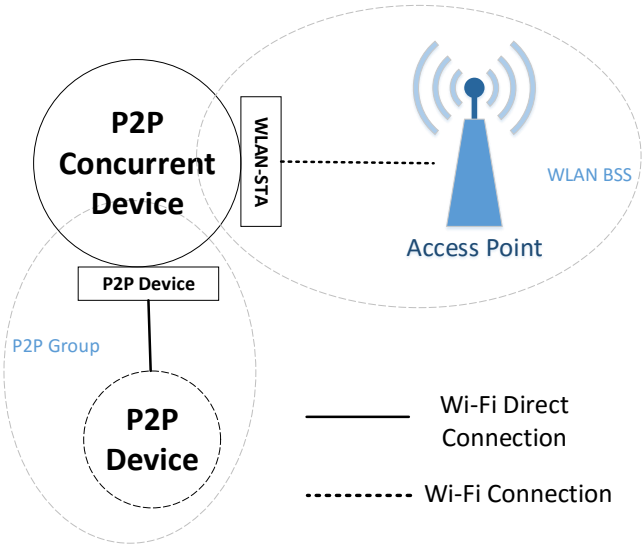


Figure 2.2: *P2P Concurrent device*

Theoretically, the maximum number of devices per group is only limited by the number of available IP addresses. However, in practice, each access point can only support tens of devices before saturating its resources. In an experiment [6] on a Cisco AP3602i access point, the maximum number of clients was between 16 and 51, based on the tested application scenario. Experiments in Wi-Fi Direct scenarios show that this number is even lower [2]. Of course, if the devices do not send and receive data frequently, less congestion occurs, and more devices can connect.

Communication As soon as a group exists, the owner activates a DHCP server and the clients activate DHCP clients. Then, the group owner can create a TCP server socket to enable communication.TCP and UDP are both allowed. It is not necessary that the group owner becomes the server. TCP/UDP connection can also be created between two clients in the same group and not necessarily between group owner and clients. However, the exact form of communication depends on the actual implementation of the protocol. While Android provides a full implementation that includes socket communication, iOS devices can also exploit the proprietary Multipeer Connectivity Framework [50] to exchange messages, resources (e.g., files), data streams with other iOS devices[3].

Power Management Power management supports power-save mechanisms for both owners and clients. This mechanism allows group owners to be absent for defined periods.

[3] In this moment, iOS does not support Wi-Fi Direct communication with other platforms like Android.

Also, it supports *opportunistic* power save that allows a owner to gain additional power savings on an opportunistic basis and supports *notice of absence* that allows owners to inform clients about the time they will be in power-save mode. Note that legacy clients are not aware of these features and any mechanism that alters the availability of the access point (i.e., the group owner) may result in undesirable consequences. Moreover, power management reduces P2P Device availability and therefore impacts the discoverability of that P2P Device [21].

2.2.2 Bluetooth Low Energy

Bluetooth is a packet-based protocol for short-range wireless communication between enabled devices. Starting with version 4.0, the Bluetooth specification introduced a light-weight version of the stack called Bluetooth Low Energy (BLE). Its main focus was low-power consumption, and it was not backward-compatible with previous versions of the protocol. BLE has increasingly gained popularity in the context of IoT and proximity-based interaction, due to the fact that it can easily be supported by very small devices. Nowadays BLE can be found on all the major OS platforms.

BLE device *discovery* is typically used in two different ways. One way is to consider discovery as a preliminary step before initiating a connection with a device.

The second way is to have a BLE device advertise small portions of information to whoever is listening in its proximity, without following this up with direct connections. Instead, the advertised information could be used by the listener to gather application-specific information –maybe from a remote cloud service. For example, an application could detect that the user has entered a room by listening to the unique id of a beacon that was previously associated with that space.

Service, or application-level, discovery is supported, but in a limited fashion. Devices can advertise one or more *profiles*, which can then be used to identify the device's behavior, its data formats, and how to create an application-level connection. To favor interoperability between different devices and applications, the specification provides a list of standard profiles [4], yet new ones can also be created.

BLE-connected devices follow a *piconet* structure [91]. One device plays the role of *central (master)*, and can accept up to seven other devices called *peripheral (slaves)*. Each slave can only send and receive data to and from the master, and the master is responsible for preventing collisions by defining time slots for communicating with the different slaves. For example, a smartphone can play the role of the master by connecting to different BLE sensors acting as slaves. The smartphone is a client when it gets data from the sensors, but it becomes a server when it sends commands and updates to them.

Several piconets may coexist within the same area, at the same time, with minimal interference. A device that is interested in connecting to another device is called an *initiator*; it starts by sending a connection request. After a slave has accepted the request, a connection can be established, and the initiator becomes the master of the piconet.

Once a connection is established, BLE data *attributes* can be exchanged using the GATT (Generic ATTribute) architecture over the Attribute (ATT) protocol transport. A GATT server receives requests from clients, and responds. It is responsible for storing

[4] https://developer.bluetooth.org/TechnologyOverview/Pages/Profiles.aspx

and making user data attributes available to all the networked clients. Every BLE network needs to have at least one GATT server. However, the GATT server and client roles are independent of the piconet's master and slaves.

Each GATT attribute is identified by a Unique Identifier (UUID), and is organized into *characteristics* and *services*. A characteristic represents a single value that the client will be interested in reading or writing. Descriptors can be associated with characteristics to provide additional information (e.g. the unit of measurement begin used or a maximum value). Services, on the other hand, are collections of characteristics. For example a Heart Rate service might have three characteristics: a Heart Rate Measurement, a Body Sensor Location, and a Heart Rate Control Point so that the information can be logged to a control behavior server. A GATT server contains all the available services that can be discovered by a connected device. Figure 2.3 illustrates the data hierarchy introduced by GATT.

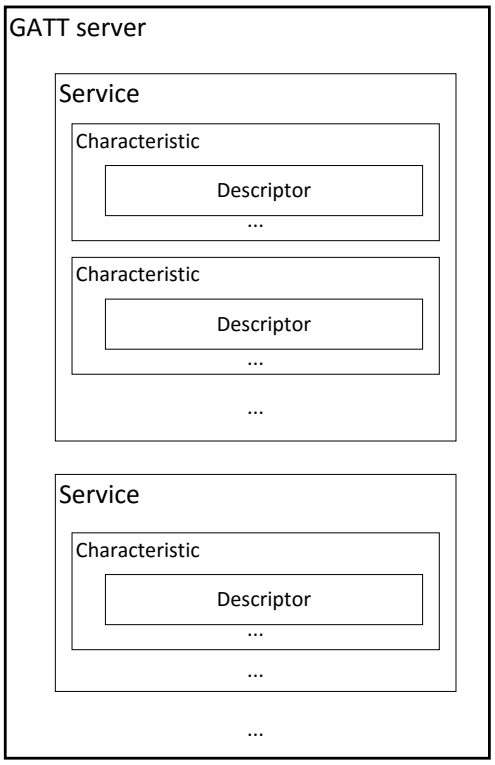


Figure 2.3: *GATT data hierarchy [82]*

Finally, a Bluetooth network can increase the number of nodes in a master/slave structure by forming a *scatternet* [3]. This allows us to remove the limit of connected nodes, which is eight in a regular Piconet (i.e., one master and seven slaves). This is achieved by having some of the slaves participate in more than one piconet. However, forming the Scatternet may reduce performance as shown in [22,28,58,91]. Moreover, a Scatternet does not imply any network routing capability [31], which would need to be provided by higher-level layers.

2.2.3 LTE-Direct

The LTE (Long-Term Evolution) standard was developed by the 3GPP consortium as a wireless high-speed mobile communication solution. Starting from release 12 it supports proximity-based discovery and device-to-device (D2D) communication under the name of LTE-Direct. LTE-Direct operates in a licensed spectrum, allowing it to provide high communication performance and reliability. Unlike Wi-Fi Direct and BLE, proximity-based discovery and D2D communication are mediated by the mobile operator, which authorizes and controls the interactions. In other words, LTE Direct is a network-assisted D2D communication technology [55, 89]. Compass.to [5] is the first application to ever showcase the features and potential of LTE Direct. It was presented at the Qualcomm booth at CES 2015 Las Vegas [8]. Moreover, Qualcomm has conducted various technical trials in cooperation with Deutsche Telekom and Huawei. These trials were aimed at testing the discovery performance of LTE Direct on Qualcomm prototype devices using Huawei's infrastructure [51].

Advertising and *discovery* in LTE-Direct focus on proximity services (also called ProSe), rather than on devices. Mobile applications can use LTE-Direct to announce their own application-specific services, as well as discover services provided by others. The discovery mechanism is highly battery-efficient and can be continuously active in the background. It is also highly scalable and can support some 1000 devices in a 500 meter range [51].

Discovery is controlled by the mobile operator through what is called a base station or cell tower (Fig. 2.4). To initiate proximal discovery an authorized application must send an advertisement request to the mobile operator's base station. Discovery is based on the notion of *expressions*: they act as filters that allow an application to monitor, and be notified by, services that satisfy the specified condition. An expression can be the application's id, or it can exploit application-dependent information, such as identity, location, a topic of interest (e.g. photography, sushi bars, etc.), or the content of a social network profile. Expressions can be private, and, therefore, discoverable only by certain users and applications, or public with the aim of reaching all devices. Expressions are represented by 128 bit codes, and managed by the Expression Name Server (ENS), which may also contain additional metadata regarding the associated service.

Once a service has been discovered, direct communication is pursued between the service provider device and its clients. The goal of device-to-device (D2D) communication is to leverage device proximity to reduce the traffic from the devices to the mobile operator's base station. Direct communication after discovery in LTE-Direct is still under development. Device to device communication for commercial applications has not been standardized in 3GPP yet. For now, only communication for public safety applications has been standardized in 3GPP release V12.0.1 [18]. This kind of communication, which is performed by broadcasting or group-casting to the proximal devices, is available both in in-coverage and out-of-coverage scenarios. These two scenario's are explained below.

Two scenarios are provided to support D2D communication: *in-coverage*, and *out-of-coverage/ partial-coverage*(Fig. 2.5). The in-coverage scenario is activated when both devices can be reached by single or multi LTE cell tower(s). This scenario supports communication for both commercial and public safety applications. The cell

[5] http://www.compass.to/

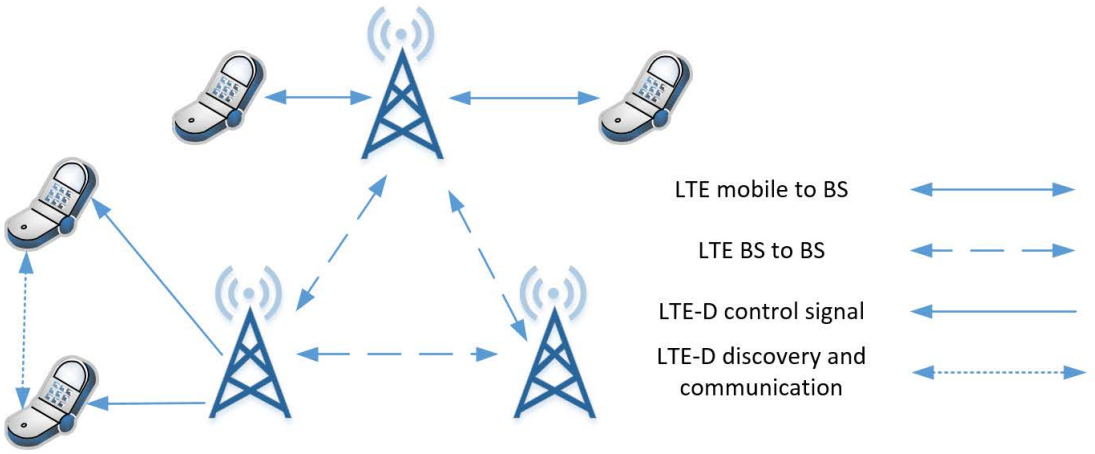


Figure 2.4: *LTE and LTE Direct network topology*

tower has the role of timing, radio resource allocation (to LTE Direct), as well as user authentication.

The out-of-coverage (and partial-coverage) scenario is only allowed for public safety applications. It supports non-mediated D2D communication, meaning that the LTE cell tower does not need to be available. Therefore, it is suitable for scenarios in which the mobile network is disrupted.

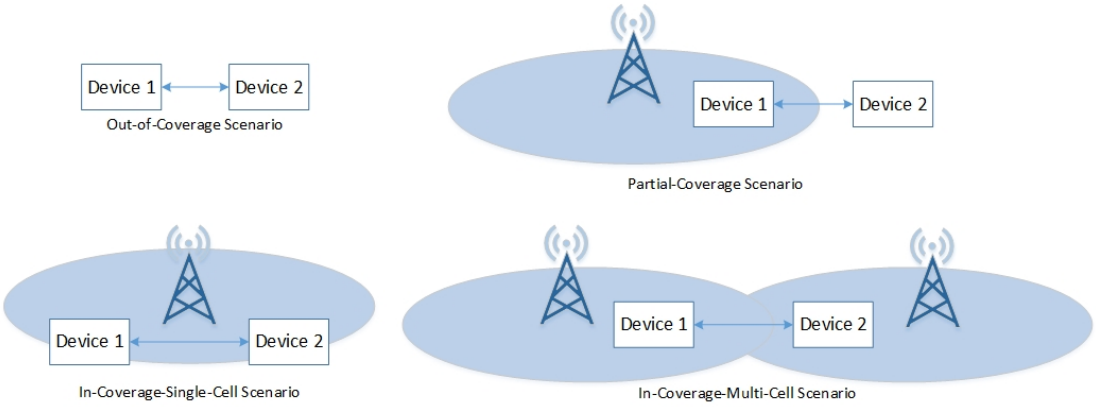


Figure 2.5: *LTE D2D Scenarios [18]*

Qualcomm provided LTE Direct trial SDK package for Android and iOS. This LTE Direct SDK Test packaging provides support for developing Android and iOS proximity apps using LTE-Direct over BLE. Android or iOS Apps will communicate with LTE-Direct enabled Dongle through Bluetooth technology [16]. This dongle is an Android Kitkat handsets with LTE-Direct capable radios.

2.2.4 Thread and 6LowPAN

Thread [17] is a self-healing mesh networking protocol that targets smart home applications; it was introduced by Google's Nest Labs in 2014. This technology is based on the 6LowPAN communication protocol, and employs the IEEE 802.15.4 communication standard, which was specifically designed for low-rate, low-powered Wireless Personal

Area Networks. Thread employs UDP over IPv6, meaning devices can communicate with one another, access services in the cloud, or interact with the user through Thread-based mobile applications. Finally, distance-vector routing is employed. Fig. 2.6 illustrates the Thread stack.

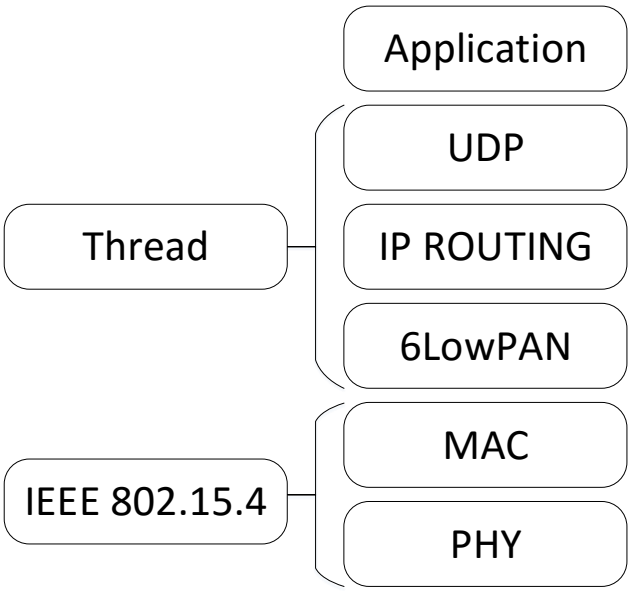


Figure 2.6: *Basic Thread Communication Stack [17]*

Thread-enabled devices can form a mesh network. To be able to self-heal and provide reliable communication, these devices should play different roles as shown in Fig. 2.7. These roles are: End Devices –also called **Leaf** or child devices–, parents –also called **Routers**–, **Border-Routers**, **Reeds**, and the **Leader**. Different roles have different responsibilities in performing the self-healing, connecting the network to the Internet or another network or routing the package to the destination. These roles are explained below.

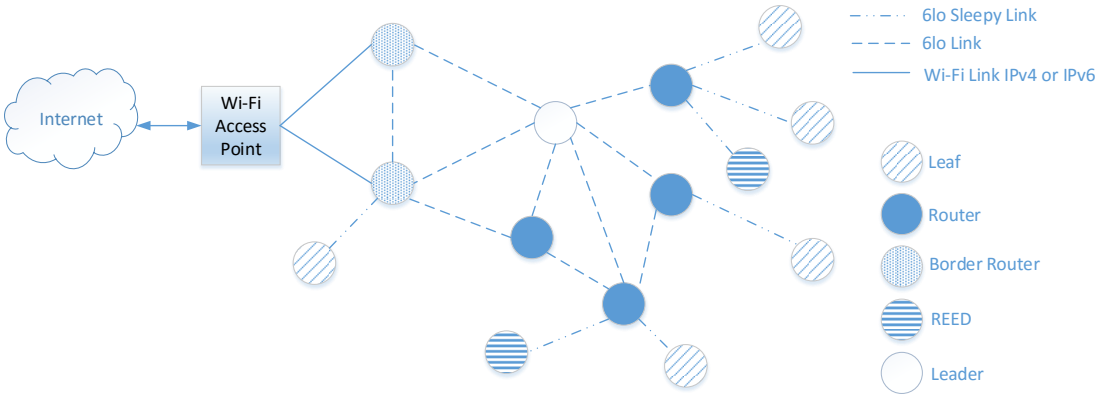


Figure 2.7: *Basic Thread Network topology [17]*

Leaf devices are battery-critical devices that do not participate in routing and package forwarding. Indeed, they are asleep most of their life, and they only wake to send

and event or to (periodically) check whether there is an event for them. Events for sleeping devices are momentarily held by the end device's parent.

Routers are always-on devices that are responsible for providing routing services to network devices. There is a limit of 32 active routers, in order to reduce bandwidth and RAM consumption [17].

A router can also be a **Border Router**, and therefore provide connectivity from 6LowPAN networks to adjacent networks using other physical layers, such as Wi-Fi.

When needed routers can downgrade their functionality and become **REEDs** (Router-eligible End Devices); this might be necessary if the number of available router devices increases. REEDS can resume their router capabilities if other active routers leave the network or fail; therefore, they can be considered to be backup routers. They behave as true end devices, but also listen to routing messages; so that when they go back online as routers they are up-to-date.

In a Thread network, one router plays the role of a **leader**. This Leader is required to make decisions for the network. For example, the leader assigns router addresses and manages new router requests.

In a Thread network roles and functionalities are defined so that there is no single point of failure. If a leader fails, or the network becomes fragmented, another router or border router autonomously becomes the new leader. If a router leaves the network, a REED will substitute the router, and the other routers will update their routing tables accordingly. Fig. 2.7 illustrates the network topology in Thread.

2.3 Middleware Technologies

This section provides an overview of the different middleware infrastructures that can be used to develop proximity-based applications. These middleware infrastructures are either industry-driven, such as AllJoyn, IoTivity, and Google Nearby, or solutions provided in the literature. At the end of each section we provide a brief summary and explain what aspects our proposed solution differ from the mentioned middleware infrastructure.

2.3.1 AllJoyn

AllJoyn is an open-source project that is developed by the AllSeen Alliance[6]. It is a platform-neutral software development framework that provides an environment for sharing resources and services across different transport layers, platforms, and operating systems. AllJoyn enables proximity service discovery, security, pairing, and message delivery. It can be run both on powerful devices, such as personal computers and mobile phones and on low-memory devices such as Arduino boards.

AllJoyn's architecture is a backward-compatible extension of D-Bus [37], the Linux inter-application communication protocol. This architecture favors interoperability by abstracting the communication to a level that is independent of the adopted network layer. To communicate over an AllJoyn bus, each device must run a daemon that acts as a local router and namespace service (see Fig. 2.8). Thanks to the daemon one can deliver messages to other daemons running on other devices, as well as to services that

[6] https://allseenalliance.org/.

are being run on that same device. To connect to the daemon, each application creates a unique *bus attachment*.

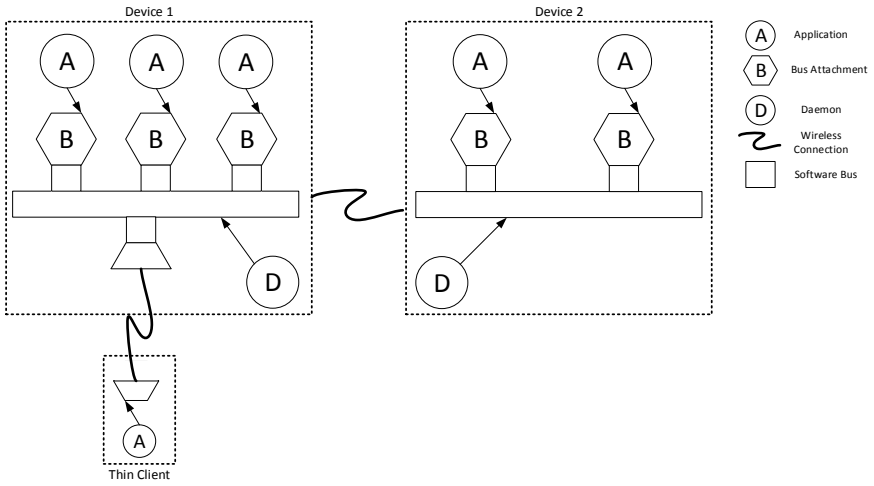


Figure 2.8: *AllJoyn Bus Architecture*

AllJoyn supports low-memory devices through the notion of a *thin client*. In this case the device does not run a local daemon; instead, the thin client must rely on a daemon that is running on some other node. This introduces some limitations. The thin client is typically run on an embedded device with a single thread, making it difficult to achieve online device detection, and to react to the device going offline abnormally.

Discovery in AllJoyn is exclusively performed at the service level, i.e., it allows us to discover the services that each AllJoyn application is offering through the bus. An application that wants to advertise its presence through AllJoyn starts by creating a bus attachment. This results in the creation of a unique identifier (UID) for that particular application. The advertisement is then achieved by registering an *about announcement* with the framework. This announcement provides a consistent set of metadata about the application that is being advertised, and can contain information such as the application's name, its version, its supported devices, and the interfaces it provides.

Clients discover applications by receiving the about announcements, after which connections can be established. AllJoyn also provides a mechanism through which one can be notified by only those applications that are deemed of interest. This is achieved by creating an *observer* object on the bus attachment that contains the interfaces to be discovered.

Application-to-application communication is enabled by *BusObject*s. These are abstractions that are registered with the bus attachment, and that expose interfaces for interaction. A BusObject interface can contain methods, properties, and signals. AllJoyn is fundamentally object-oriented. BusMethods allow a remote entity to call a method on a BusObject; BusProperties consist of data that are contained within the BusObject, and that can be remotely set and read; finally, BusSignals are AllJoyn's means to enable asynchronous notifications of event or state changes in a BusObject.

Once two applications have completed the discovery phase, they can initiate communication by creating a session. Sessions can be point-to-point or multipoint to support group communication. Once a connection has been established, the client application

creates a *ProxyObject* to interact with the corresponding remote *BusObject*. The interaction is then achieved using Remote Procedure Calls.

Although AllJoyn offers interesting solutions for networking, mobility, security, and dynamic configuration, it does have some weaknesses. First of all, it lacks scalability; in fact, it does not support communication between devices that belong to different broadcast domains (i.e., belonging to different subnets). Second, AllJoyn cannot manage the case in which the number of devices on a Wi-Fi Direct network exceeds the group's capacity. Moreover, it does not provide any real-time analytics capabilities. An enormous amount of data may be generated and shared through the bus when the number of connected devices and/or the information acquisition frequency increases. This can make data management very hard to achieve [85]. Last, AllJoyn could be used on top of Wi-Fi Direct, but it does not provide any mechanism for autonomous group formation or failure management. User intervention is necessary to reconnect the failed device to a Wi-Fi Direct network. This is where MAGNET contributes innovative solutions.

2.3.2 IoTivity

IoTivity [44] is an open-source software framework sponsored by OIC, the Open Interconnect Consortium[7]. It provides a device-to-device communication layer for the Internet Of Things. The project is still in an early phase. Nevertheless, it already provides important features such as discovery, connectivity, security, resource management, group management, and plugin extensions.

The framework provides a communication infrastructure for devices with limited capabilities, called *constrained* or *lite* devices, as well as for more traditional devices such as smartphones, tablets, and laptops, generically called *unconstrained* or *rich* devices. Similarly to AllJoyn, lite devices in IoTivity rely on rich devices for operation.

IoTivity utilizes a distributed Bus architecture, called the *Base*, to provide both communications between devices, and communication between applications that reside on a single device. The Base provides Messaging (CoAP over TCP), Security (DTLS), Multicast Discovery, and Connectivity abstractions (Wi-Fi, BLE, Bluetooth abstraction with CoAP). All IoTivity devices are participating in the distributed implementation of the Base.

The Base provides C-based APIs for applications on Lite devices and a richer service layer for Rich devices. The service layer provides features such as resource management (group control and device configuration), data management (sensor management and message translation between different protocols), and low-power management. The service layer provides Java, C++, and Web APIs for end-user applications. Moreover, it offers a plugin for non-OIC devices, so that they can communicate with OIC devices [36]. Fig. 2.9 illustrates IoTivity's general architecture.

Devices are logically represented as *resources*; they can be discovered, and they can interact with other constrained or unconstrained devices. Resources are organized in a hierarchical tree, both to group similar resources and to define their management. Indeed, parent resources are given the capability of controlling their children resources. As an example, a health monitor system might be a parent resource while its different

[7] `http://openinterconnect.org/`.

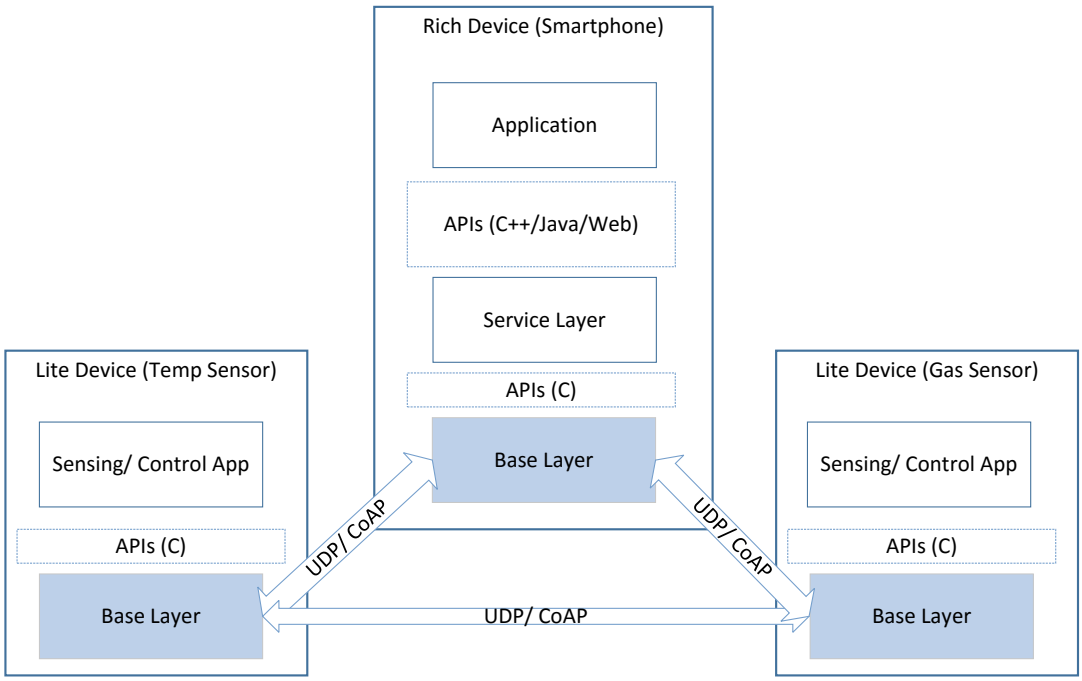


Figure 2.9: *IoTivity Architecture*

sensors would be its children. Resources can contain attributes that are values that other devices can remotely read or update.

Resources are organized following a REST-like architectural pattern. Each resource is identified by a URI that follows the CoRE (Constrained RESTful Environments) format [8]. Resources interact with each other by following the OIC protocol, a protocol that is similar to HTTP and CoAP (Constrained Application Protocol [9]).

An application that wants to advertise a resource needs to register it within the framework. Once a resource has been registered, other devices can discover it through a procedure that operates by sending a multicast GET request that specifies the resource's type. Resources belonging to that type will send an acknowledgment back to the caller.

Once a resource has been discovered, two main types of interaction become possible. On the one hand, the application can read (through a GET request) or update (through a PUT request) the resource state. On the other, resources can be defined as observable. In this case, client applications must declare their interest in observing the state of the resource; as a result, they will be notified if the resource's state is updated.

Fig. 2.10 shows an example scenario for IoTivity in which a Soft Sensor Manager (SSM) is provided in the service layer's Data Management module. A soft sensor is an abstraction of a physical sensor that is responsible for collecting and manipulating sensor data for applications [54]. The SSM provides a service through which one can submit query statements about soft sensor data (e.g., temperature, humidity, or gyro sensors); the results of these queries are provided through the IoTivity Base using Context Query Language (CQL) [14].

[8] https://datatracker.ietf.org/doc/rfc6690/
[9] http://datatracker.ietf.org/doc/rfc7252/

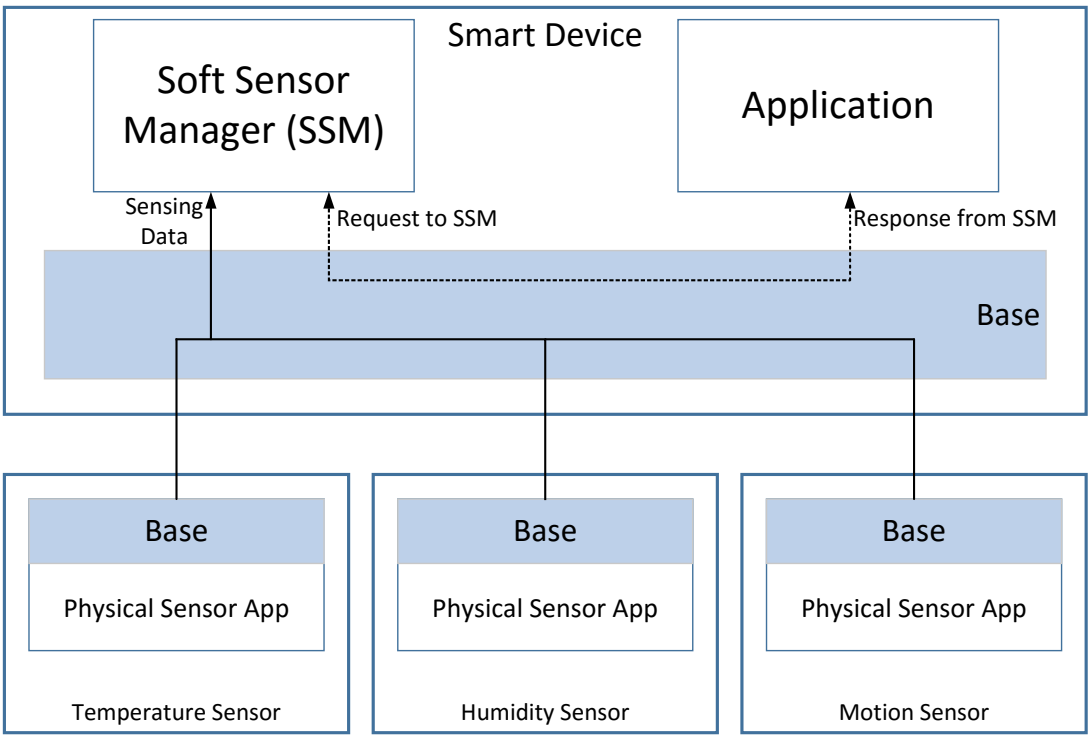


Figure 2.10: *SSM Context Diagram [54]*

Similar to AllJoyn, IoTivity acts as an enabler for the seamless device-to-device connectivity. The current release of IoTivity supports only Wi-Fi and Bluetooth, but in theory, it could also exploit Wi-Fi Direct. However, it does not provide any mechanism for autonomous group formation or failure management. Moreover, it cannot manage the case in which the number of devices on a Wi-Fi Direct network exceeds the group's capacity. This is where, together with the smart management of failures, MAGNET contributes innovative solutions.

2.3.3 Google Nearby

Nearby[10] is a publish/ subscribe middleware technology for Android and iOS platforms which is introduced by Google in 2015. The aim of this technology is to build simple, interoperable interactions between nearby devices. Applications can share messages and create real-time connections between nearby devices by using two distinct API: `Nearby Message API` and `Nearby Connection API`.

Nearby Message API

With the help of `Nearby Message API`, the proximal Android or iOS devices, which are connected to the Internet but not necessary are on the same network, can exchange small payloads[11] of data utilizing publish/ subscribe paradigm. A publishing application should first make a request to the message server to associate the message

[10] `https://developers.google.com/nearby/`

[11] Although the Maximum size of the payload is limited to 100 KB, Google recommended to keep the size below 3 KB to maintain good performance

with a unique-in-time pairing code called *token*. After the token is associated, the publishing device uses a combination of Bluetooth, BLE, Wi-Fi and near-ultrasonic (inaudible) audio [63] to make the token detectable by nearby devices. A subscribing application should also associate its subscription with a token and uses a mix of the above technologies to send its token to the publisher, and to detect the publisher's token. After a token is detected by either mentioned technologies, the application should report it to the server. The server facilitates message exchange if the tokens of two devices are the same and the API keys used by the calling applications are associated with the same project in the Google Developers Console[12]. Moreover, using the `Nearby Message API`, an application can subscribe to BLE beacon messages using the same mechanism that is used to subscribe to messages published by other nearby devices. Fig.2.11 illustrates the above message exchange mechanism which is being employed by the `Nearby message API`.

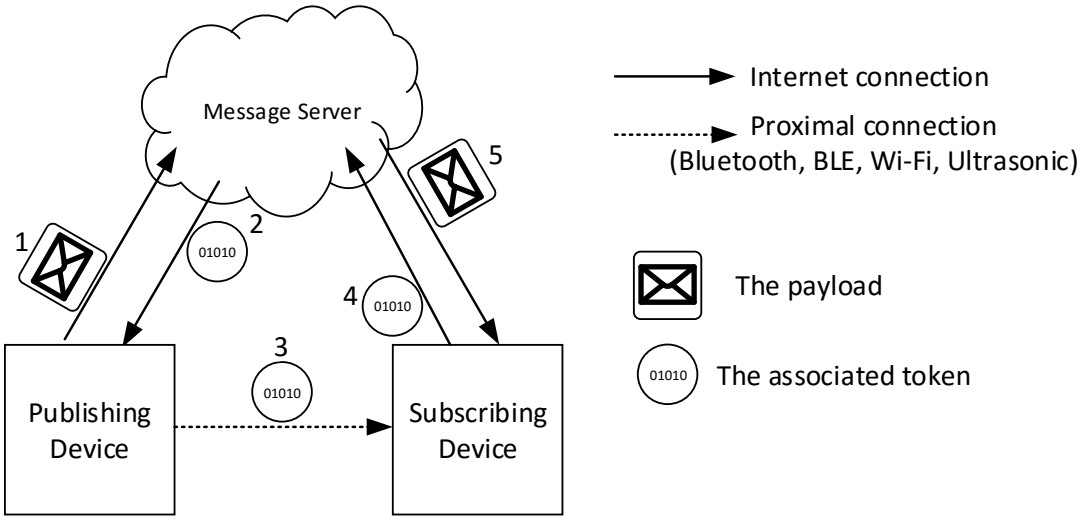


Figure 2.11: *Nearby Message API overview*

The developer can only call publish() and subscribe() methods. The token association, advertisement, and discovery are transparent from the developer point of view. By default, token broadcasting and scanning is done on all the available mediums in a device. However, it is possible for an application to control the set of mediums used for device discovery, and whether the mediums are used to broadcast tokens and/or scan for tokens. Because of high usage of radios and sensors in Nearby message communication, an active publish/ subscribe will cause an application to consume battery up to three and half times the normal rate [11].

Nearby Connection API

`Nearby Connection API` provides an abstraction of discovery and communication on a local network (Wi-Fi or Ethernet) for Android devices. To establish a connection between two or more devices, at least one device, which is called *host* and has a service to offer, advertise a uniquely identified `service_ID`. Then, other devices in the same network can discover nearby applications that are advertising with

[12] `https://console.developers.google.com`

the required `service_ID` and connect to them as clients. It is like a user that starts a game and become a host for the other parties to participate in the game as clients. Once the devices are connected, they can send and receive messages to update application state and exchange input, data, or events between devices. In contrast to the `Nearby Message API` that the payloads exchanges facilitated by the server on the cloud, the message communication in `Nearby Connection API` is handled in real-time using a multicast-enabled local network and there is not any limit for the size of the message. This API is particularly interesting in *Collaborative white-board*, *Local multi-player gaming*, and *Multi-screen gaming* application domains.

Although Google Nearby enables simple proximity-based interactions between nearby devices and people, the current release does not support Wi-Fi Direct. Moreover, its intrinsic need for an active Internet connection may prevent the exploitation of Google Nearby framework in some significant IoT scenarios where the Internet connection is not available.

2.3.4 Prototypal Solutions

In addition to these industry-supplied solutions, many other prototypal academic solutions enable P2P social interactions between proximal devices. In this subsection, we briefly discuss similar solutions to MAGNET.

To the best of our knowledge, MAGNET is the first middleware infrastructure that provides reliable, multi-hop, and ad-hoc communication among a large number of Wi-Fi Direct-based mobile devices. However, other, similar solutions address infrastructureless networking and exploit widely available P2P wireless communication protocols, like Wi-Fi ad-hoc, Wi-Fi Direct, and Bluetooth. Bellavista et al. [29, 30] present a middleware infrastructure called **MMHC** for the multi-hop, multi-path heterogeneous communication in mobile environments using Wi-Fi ad-hoc and Bluetooth. Their main goal is to extend Internet connectivity to dynamic environments. They do not provide any specific mechanism to support large numbers of devices. For instance, they do not clarify how they manage the case in which a Bluetooth piconet[13] reaches its maximum slave capacity. As for Wi-Fi ad-hoc, it has proven to be very unreliable in the presence of a vast number of devices [86]. It is also very energy-hungry [66] and this makes its utilization infeasible with battery-powered devices like smartphones.

Some other works have tried to provide a P2P network based on Wi-Fi Direct. Santos et al. [75] propose My-Direct, which uses Wi-Fi Direct together with Bluetooth to support flexible communication among the nodes of a mobile social network. However, their work does not support failing group owners or clients and also it is not compatible with a significant number of devices. The middleware does not provide any self-adaptive capability, and human intervention is needed to reestablish connections with proximal devices. Chaki et al. [35] solve the problem of failing group owners. They assume that groups have already been created. Group members create persistent group configurations artificially, that is, configurations without any actual group formation, and elect a client as a backup to become owner if needed. These configurations are then exploited[14], if needed, through persistent group formation (Section 3.2). Although this

[13] A piconet is a network of Bluetooth devices. The specification says that up to eight devices can be connected. One device acts as a master (server) and can host up to seven other devices as slaves (clients).

[14] This solution has been evaluated on Linux, which provides better access to the kernel. Since Android does not provide any

solution can manage group failures autonomously, it does not address group saturation, and thus it is not suitable when facing large numbers of devices. Group bootstrapping, client failures, group saturation, and inter groups communication are not discussed in this work. Table 2.1 compare the aforementioned middleware technologies regarding mobility and large-scale support.

Table 2.1: *A comparison in terms of mobility and large-scale support*

	AllJoyn	IoTivity	Nearby	MMHC	MyDirect	Chaki	MAGNET
Wi-Fi Direct Support	✓	✓	✗	✗	✓	✓	✓
Device Failure Management	✗	✗	✗	✓	✗	✓	✓
Device Mobility Support	✗	✗	✓	✓	✗	✓	✓
Group Saturation[a]	✗	✗	✗	✗	✗	✗	✓
Multi-hop Routing	✗	✗	✗	✓	✗	✗	✓
Inter Group Communication[b]	✗	✗	✗	✓	✗	✗	✓
Device Discovery	✗	✓	✓	✗	✓	✗	✓
Service Discovery	✓	✓	✓	✗	✗	✗	✓

[a] Wi-Fi Direct group saturation management
[b] Between Wi-Fi Direct groups

2.4 Discussion and Comparison

This section proposes a first comparison of the different industry-driven technologies described above and discusses their suitability for various application scenarios. As for the comparison, we have taken into account how the different technologies support device/application/service/resource advertisement and discovery, their capability to scale to support high quantities of proximal devices, and the extent to which they support different development platforms and devices (i.e., what operating systems and programming languages they currently support). As a result, we attempt to clarify in which application scenarios the different communication protocols and middleware infrastructures are suitable. A summary of the comparison is provided in Table 2.2.

When discussing the discovery of proximal peers, we should distinguish between device and service discovery.

Device discovery is useful in applications in which the devices that might be available in proximity are known in advance, or when they are easily recognizable from the device's name. This goes in the context of *personal area networks*, e.g., areas involving wearable devices that can be easily identified. Proximity-based applications that need to discover unknown devices and that are built on top of technologies that only support device discovery, need to maintain a mapping between the device's id and some form of application-specific information. This mapping may be facilitated by the usage of cloud-based storage or be maintained locally on the user's device. For example, if we want to use BLE beacons to identify rooms in a building, we will need to associate the

API to support persistent group configuration manually, root access would be needed, but this is not safe and suggested in Android.

rooms with the beacons' UUIDs. This way when the user's mobile device comes in contact with the beacon, it will be able to infer in what room it is.

Service discovery, on the other hand, can have two different uses. It can be seen as an application discovery mechanism, that is, as a way to expose and discover application-level functionality, or as a resource discovery mechanism, that is, as a way to describe the information provided by an advertising device, as well as to clarify how they can be accessed.

Another important aspect to take into account when developing a proximity-based application is the moment in which the discovery has to happen. Discovery can be performed either before or after connecting to a device. Wi-Fi Direct and LTE-Direct can provide service information before the connection is made. This is especially useful when the devices are not known a priori, and they can be filtered directly by the application before establishing a connection. BLE, Thread, AllJoyn, and IoTivity, on the other hand, require that the devices be connected before discovering their capabilities. In Nearby Message API, the devices only discover the tokens and the message communication is done over the Internet.

Depending on the application scenario being developed, scalability may play a significant role in choosing one technology over another. There are two ways to evaluate scalability. One can measure the maximum number of devices that can be discovered and connected, as well as evaluate the topology that is created once they are connected.

As for the maximum number of connectable devices, BLE and Wi-Fi direct offer limited scalability, and are therefore more suitable for small areas. Bluetooth can only support eight devices in a Piconet, and although this could be extended using Scatternets, this solution is not currently part of the Bluetooth specification. As for Wi-Fi Direct, there is no theoretical maximum number of connectable devices in the standard. Limitations are actually introduced by the saturation of the AP. The highest number of devices that an AP can host is between 10 to 51 [6], although another experiment [7] resulted in 56 simultaneous connections under low-traffic load conditions. With Wi-Fi Direct, this number becomes even smaller due to the group owner's power consumption. Communication in LTE Direct is still under development. For now, only public-safety applications can communicate by broadcasting messages. Since broadcasting does not need a prior connection, this comparison is meaningless for LTE Direct. A Thread network can host up to 300 devices simultaneously with the use of 32 routers [81].

AllJoyn and IoTivity, on the other hand, use Bluetooth and Wi-Fi Direct as their underlying communication technologies. This means they are also bound to the limitations mentioned above. Neither AllJoyn nor IoTivity is scalable; they cannot cross-connect Wi-Fi Direct groups or Bluetooth Piconets to form a bigger network. However, if the underlying communication channel chosen for these middleware is traditional Wi-Fi, the number of devices could be expanded by adding additional access points to the network.

Wi-Fi Direct can be a good choice whenever high-speed P2P communication is needed. Wi-Fi Direct provides an open-space communication range of more than 200 meters (at regular Wi-Fi speeds), and its battery drain is limited with respect to traditional Wi-Fi networks. BLE and Thread, on the other hand, have lower ranges and lower speeds and are suitable when energy consumption is a primary concern. With

LTE-Direct, one can theoretically discover thousands of devices in an area of 500 meters. However, since the technology is not widely available yet, the actual scalability of device-to-device communication is still open for future evaluation. MAGNET is built on top of Wi-Fi Direct instead of BLE or Thread. The main reason for this is the low range of Bluetooth and 6LowPan technologies, which severely limits the functionality of MAGNET low-density situations, and the devices are not necessarily very close to each other. Moreover, in high mobility scenarios, lower radio range will result in higher frequency of connection and disconnection, which decreases the stability of the network.

The network topology must be evaluated with respect to its capability of addressing application scenarios in mobility, and its ability to dynamically adapt to changes in the proximal peers. Among the different technologies we have presented, LTE-Direct seems to be the most promising in this sense, given the coordination role of the base station. Wi-Fi Direct follows a star topology, with one device playing the role of the group owner and several devices behaving as clients. Thread is a mesh network in which devices can talk with one another either directly or via multi-hop communication. Bluetooth devices can form Piconets in which one device plays the role of the master, and up to seven devices play the role of slaves. AllJoyn is a star-bus. This means we do not need to place every packet on the bus; if the packet destination is on the local subnet, we can deliver it directly. This lead to lower congestion; with Bluetooth and Wi-Fi Direct every packet needs to be transferred either to the group owner or to the master. Moreover, if we have several applications running on the same device, they all need to communicate through the group owner or the master, increasing congestion even more. Finally, IoTivity follows a distributed bus architectures.

The actual realization of proximity-based applications also depends on the development support provided by these technologies, both regarding the operating systems and the programming languages they support. Wi-Fi Direct is supported by almost all of the leading operating systems, although with some limitations as already discussed for iOS. Although LTE Direct is still under development, initial support for Android and iOS is already available. AllJoyn and IoTivity support all popular operating systems. The thin client in AllJoyn and the lite client in IoTivity even allow us to support embedded platforms, such as Arduino. Nearby is available for Android and iOS platforms

Another important factor is how easy it is to debug an application that uses one of these technologies. How easy it is to debug a specific technology can be gauged as the extent to which appropriate simulation or emulation platforms are available.

There are many simulators available for traditional Bluetooth. For instance, IBM provides BlueHoc [52], an open source Bluetooth stack for the NS-2 simulator. Godfrey Tan also provides a Bluetooth extension for NS-2 called BlueWare [80]; its main peculiarity is that also supports scatternet network formation. Other well-known Bluetooth simulators are [26] and [57]. BlueSim [47], on the other hand, focuses on simulating BLE peripherals (e.g. heart rate or blood pressure sensors) for iOS devices.

The only publicly available simulator for Wi-Fi Direct networks is WiDiSi [27, 38], a research-oriented prototype simulator provided as an extension of PeerSim [59]. [73], which will be explained in detail in Chapter 3. To the best of our knowledge, WiDiSi is the first Wi-Fi Direct simulator that supports device and service discovery, standard and autonomous group formation, and communication between devices. It is also the only

simulator written in Java that provides the same API as the one provided by Android.

As for IoTivity, official simulators have been provided by OCI. IoTivity Simulator is an Eclipse plugin tool. It offers two perspectives: 1) Service Provider that manages the creation, deletion, request handling and notifications of simulated resources and handles the requests received and sending appropriate responses to clients, and 2) Client Controller that simulates the functionality of OIC client. It can find resources of interested types in the given network and provide support for sending automatic requests (GET/PUT/POST) to remote resources [76].

No simulation/emulation platform has been introduced for Nearby, AllJoyn, Thread, and LTE Direct yet.

Table 2.2: *Comparison in a nutshell*

	Bluetooth LE	Wi-Fi Direct	LTE Direct	Thread	AllJoyn	IoTivity	Google Nearby
Type	Open Standard-/API	Open Standard-/API	Open Standard-/API	Open Standard-/API	Middleware	Middleware	Middleware
Network Topology	Piconet/ Scatternet	Star	Network-assisted P2P	Mesh	Star-Bus	Bus	Cloud-based and P2P
Device Discovery	✓	✓	✗	✓	✗	✓	✗
Application Discovery	✗	✓	✓	✗	✓	∼	✓
Resource Discovery	✓	✓	✓	✗	✓	✓	✗
Max No. of discoverable devices	Low	Low	High	N/A[a]	N/A[a]	N/A[a]	Low
Max No. of connected devices	Low	Low	∼	High	High	High	Low
Discovery Speed	Slow	Slow	Fast	N/A[a]	Fast	Fast	Slow
Supported Communication Protocols[c]	-	-	-	6LowPAN	Wi-Fi, Ethernet, Serial, PLC	CoAP	Wi-Fi, Bluetooth, BLE, Ultrasonic, Ethernet
Supported OS	Linux, Android, Win, iOS, OS X	Linux, Win, Android, iOS[d]	Android, iOS	N/A[a]	RTOS, Arduino, Linux, Android, OS X, iOS, Win	Linux, Android, Arduino, Tizen, Yocto	Android, iOS
Language Bindings	All	C, C++, Objective-C, Java	Java, Objective C	C , C++	C, C++, Objective-C, Java	C, C++, Java, JavaScript	Java, Objective-C
Suitable For:							
Embedded Devices[b]	✓	✗	∼	✓	✓	✓	✗
Low-Power Devices	✓	✗	✓	✓	∼	✓	✗
Very High Speed P2P	✗	✓	✗	✗	✓	✗	✓[f]
Mobility Scenarios	✗	✗	✓	✗	∼	∼	✗
Marketing	✓	✗	✓	✗	✓	✗	✓
Social Interaction	✗	✓	✓	✗	✓	✗	✓
Internet of Things	✓	✓	∼	✓	✓	✓	✓
Open Source	✗	✗	✗	✗	✓	✓	✓
Simulator availability	✓	✓	✗	✗	✗	✓	✗

[a] Not available
[b] Low-Power, Low-Processing capability
[c] Only protocols that currently are being supported
[d] Only iOS to iOS is supported
[d] Only in Nearby Connection API which uses Wi-Fi or Ethernet for communication

CHAPTER 3

Wi-Fi Direct Simulation Environment

This chapter introduces WiDiSi, the first Wi-Fi Direct simulator that supports device and service discovery, standard and autonomous group formation, and communication among devices. It is also the only simulator written in Java that provides the same API as the one provided by Android. The rest of the chapter is organized as follows. Section 3.1 is problem statement and motivation. Section 3.2 gives an overview of the PeerSim simulator. Section 3.3 presents the architecture of WiDiSi. Its implementation is then presented in Section 3.4, and Section 3.5 presents the evaluation of WiDiSi.

3.1 Problem and Motivation

Since the introduction of the Wi-Fi Direct standard, many companies have started providing hardware and software implementations. The most notable one is the one provided by the Android operating system. It provides a comprehensive API for discovering, connecting, and communicating with other WiFi Direct-enabled devices and, unlike iOS, it can be utilized for communication between any Wi-Fi Direct-enabled devices. Unfortunately, the Android emulator does not support Wi-Fi Direct. As a result, the only way to test an application that uses Wi-Fi direct is to use physical devices. Moreover, even if the Android emulator were to support Wi-Fi direct in the future, its heavy use of system memory would make it virtually useless for testing large-scale applications.

Using physical devices, however, can pose many problems. The first obvious issue is that, in order to test a large-scale application, one must possess a significant number of devices. This might be very costly, or even unfeasible. On top of that, setting up the test application on every single device, and collecting the runtime data that one might want to analyze, can also be very hard to achieve. It can be very challenging to

generate a test scenario in which devices can move in arbitrary patterns, enter or leave the system at will, or fail due of battery issues. With real devices, it can also become hard to identify whether a problem is originating in the hardware or the software. For example, a powerful source of noise in the devices' proximity might jam the Wi-Fi Direct signals and cause communication disruption. Finally, replicating a real-world scenario might prove to be impossible, due to the fact that execution contexts cannot easily be replicated.

The aforementioned reasons and the need for a Wi-Fi Direct simulator in work led us to create WiDiSi. WiDiSi is, to the best of our knowledge, the first Wi-Fi Direct simulator for Android applications. It is designed to support a large number of Wi-Fi Direct-enabled nodes and user-defined mobility patterns. WiDiSi is highly configurable, and can support very dynamic situations. Indeed, a user can define complex mobility patterns, as well as simulate devices entering and/or leaving the network, or failing. Furthermore, WiDiSi provides the same WiFi Direct API that Android provides, making it easy to port an application being tested in the simulator back to real devices. Finally, WiDiSi provides a graphical interface through which the user can visualize how the large-scale simulation is proceeding.

WiDiSi is built on top of PeerSim [59], a widely-used, open-source simulation framework for large-scale peer to peer networks. Our evaluation and verification of the Wi-Fi direct simulator indicate that it correctly follows the Wi-Fi direct specification [4], and that it is capable of simulating large-scale applications.

WiDiSi is also the only simulator written in Java that provides the same API as the one provided by Android. There are some similar works that try to provide solutions to test Wi-Fi Direct applications. Bernardo [73] developed a solution to add Wi-Fi Direct support to the Android emulator and other third party emulators like Genymotion [1]. However, the use of emulators is already a bottleneck for large-scale scenarios. The solution comprises a Java-based console and a Wi-Fi P2P API. The console is responsible for managing groups (i.e., creation, join, and termination) and for providing the communication channel between emulators. It also mimics the proximity of devices by sending each application the list of claimed-to-be-proximal devices. This solution does not support service discovery and autonomous group formation. The API offers a limited set of features that narrow the use of this solution. Another distinctive feature is that most of the main Wi-Fi Direct features can only be operated through the external console, and not within the emulators.

The MOTO simulation platform [2] uses WiFi ad-hoc connection on NS-3. (NS-3 provides a complete WiFi stack over the IEEE 802.11 interface.) MOTO utilizes the Wi-Fi ad-hoc interface in NS-3 to evaluate the performance of different offloading strategies in LTE networks. Although Wi-Fi ad-hoc is different from Wi-Fi Direct, this solution can be useful for all those applications that only exploit P2P communications, and are not interested in discovery and groups.

Finally, Harri et al. [48] developed the Wi-Fi Direct interface over Wi-Fi ad-hoc communication in NS-3 as an extension to the iTETRIS platform [12]. However, the implementation details are not publicly available. Moreover, NS-3 is written in C, and they do not provide any API for Android, iOS, or any other operating system for

[1] https://www.genymotion.com/
[2] http://www.fp7-moto.eu/

mobile devices. This means that the portability between real devices and the simulator is cumbersome and time-consuming.

3.2 PeerSim

PeerSim [59] is a simulator that can simulate very large-scale peer-to-peer networks. Nodes in the network join and leave continuously. It supports both *cycle-based* simulation models and more traditional *event-based* ones. Since PeerSim mainly focuses on performance simulations at the network overlay level, it does not provide any details regarding the underlying communication network, such as the TCP/IP stack, latencies, etc. [61]. The simulator structure is based on components (i.e. Protocol, Control, Node, and Linkable) and makes it easy to quickly prototype a protocol, combining different pluggable building blocks, that are in fact Java objects.

We built our simulator on top of PeerSim, instead of using other well-known network simulators like NS-3 [9] or Jemula802 [74]. This decision was made for various reasons. First of all, PeerSim is designed for Peer-to-Peer (P2P) networks, while NS-3 or Jemula802 are general network simulators. As a result, PeerSim models the underlying network using concepts such as delay and package drop rate, while in NS-3 or Jemula802 the underlying communication channels are modeled in much greater detail. As a result, PeerSim is more lightweight and can simulate more P2P nodes than network simulators [61, 77]. Second, in PeerSim nodes contain identical behaviors, while in NS-3 or Jemula802 we have to define the behaviors that are installed on each individual node. This added flexibility may be important in certain scenarios, yet it comes with additional costs, both in terms of the effort required to implement the system, and of the efficiency of the running simulations. Wi-Fi Direct, on the other hand, must be able to cope with nodes that enter and leave the network frequently; this means that the simulator must be able to deal with these changes quickly and efficiently. This comes more naturally in PeerSim.

When developing a PeerSim simulator one must implement the Java classes that provide the behavior that we want to simulate. These classes implement a series of Java interfaces provided by PeerSim. There are four main interfaces: *Protocol*, *Control*, *Initializer*, and *Linkable*. Classes that implement the *Protocol* interface get installed onto each *Node* (a node represents a simulated device); they identify the behaviors that the nodes should manifest, once per simulation cycle or when triggered by an event. Classes that implement the *Control* interface are used to monitor and manage the status of the network. They have direct access to all the Protocol components and are instantiated once per simulation. The *Initializer* interface represents a special kind of Control. It is instantiated once per simulation, yet its code is run once per node instantiation. Finally, we have one implementation of the *Linkable* interface per node in the simulation; through this interface, each node provides programmatic access to a list of its neighbor nodes.

3.3 Proposed Simulator Architecture

Figure 3.1 illustrates WiDiSi's architecture. A Wi-Fi Direct network consists of multiple devices. These are abstracted as *Node*s in the simulator, and shown at the bottom of

the figure. The various nodes are running within a *Node Container*, which is responsible for keeping track of what nodes are within the simulation at all times.

Every node in the simulation runs a set of *Applications*. These applications represent the behaviors that the node would have on a real Android device. Each node also includes the *Wi-Fi Direct Interface*. This interface is what implements the Wi-Fi P2P protocol inside the simulator; more specifically it pretends to be the Android implementation of the specification [4].

On top of the container we have various components that are used to setup the simulation, manage it, and analyze its behavior through logging and advanced visualization. The *Node Initializer* is responsible for initializing each node when it is added to the network, while the *Proximity Manager* is in charge of defining each node's proximal elements. In fact, it contains a list of all the neighboring nodes that each node can see and connect to.

The devices in a Wi-Fi direct network can and will move; therefore, it is very unlikely that proximity relationships will remain stable over time. On top of that, nodes can also fail due to battery issues. These problems are simulated by the *Network Dynamism* component. It adds and removes nodes to the network according to a user-configurable pattern, and moves the nodes as the simulation proceeds.

The *Node Communication Manager* handles the message exchanges occurring between nodes. These messages can be related to Wi-Fi Direct or other simulator-related events.

The *Logging* component monitors how the network behaves by collecting runtime information. It has access to all the nodes in the network, and can log data for further analysis. The *Network Visualizer* uses the data collected through the logging component to provide the user with a visualization of the evolving network.

Finally, the *Scheduler* is what drives the simulation engine itself; it uses PeerSim cycles. In WiDiSi, we can arbitrarily define a duration for each *PeerSim cycle*. For instance, we may decide that one cycle lasts one millisecond, and thus one second would correspond to 1000 PeerSim cycles. All PeerSim components are synchronized around PeerSim cycles.

This architecture allows the developer of an Android application to import the WiDiSi libraries (instead of the existing Wi-Fi P2P libraries), and develop applications that are then run inside the simulator. Although we have tried to keep the API as similar as possible to the real Wi-Fi P2P API, some minor changes were necessary, and will be discussed in section 3.4.1.

3.4 Implementation Details

In this section, we will provide details on how the main components in the WiDiSi simulator were implemented [3]. We will start with the *Wi-Fi Direct Interface*; it plays a pivotal role in WiDiSi since it substitutes the real Android Wi-Fi Direct API when applications are run inside the simulator. We will discuss how we inject dynamism and deal with proximity in our simulations, and how we initialize new nodes in the system. We will examine the effects of IEEE 802.11 on the simulations, and conclude with a presentation of our simulation visualization tools.

[3] The full implementation of WiDiSi is available at `https://github.com/nasser1941/WiDiSi/`.

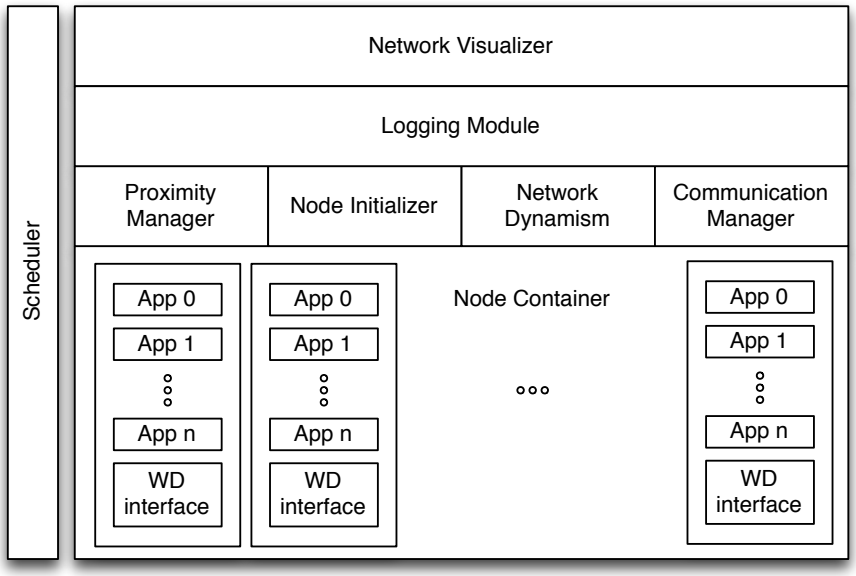


Figure 3.1: *Overview of WiDiSi*

First, however, we need to clarify that the behavior of each peer in the simulation, with respect to Wi-Fi Direct, is determined and modeled through parametric delays. The most important ones are:

- *SwitchingDelay* is caused by switching the channels during the search phase. It represents the average time that two peers take to find a common channel;
- *ChannelDelay* mimics the time needed for the physical propagation of signals. It is the amount of time needed to "successfully" exchange one frame between two peers at the MAC layer;
- *AuthenticationDelay* is the user-dependent delay for authentication. It is the amount of time needed for the user to accept an invitation and perform the provisioning phase;
- *EncryptionDelay* is additional delay caused by the AES-CCMP encryption process;
- *PowerManagementDelay* is the amount of time that a device stays in sleeping mode;
- *InternalProcessingDelay* is the time taken to process a basic action.

The last parameter takes into account the Android operating system. This is important because activity scheduling within the operating system causes overheads and delays that must be added to the time required to complete any action in a Wi-Fi direct related operation.

3.4.1 Wi-Fi Direct Interface

The *Wi-Fi direct interface* is at the core of our Wi-Fi Direct simulator. It consists of a set of classes that replicate the behavior of Android's own Wi-Fi Direct implemen-

tation. More precisely, these classes implement PeerSim Protocols. Their names are *WifiP2pManager*, *NodeP2pInfo*, and *EventListener* (see Figure 3.2). The first replicates the behavior of Android's own `android.net.wifi.p2p.WifiP2pManager` [13]. This class provides the API for managing Wi-Fi peer-to-peer connectivity. With the help of this class, an application can discover available peers, set up connections and query for the list of peers. We tried to keep the behavior of this API as close as possible to the one proposed by Android. Unfortunately, this was not entirely possible, and as a result, we do not currently support all the features made available by Android's implementation.

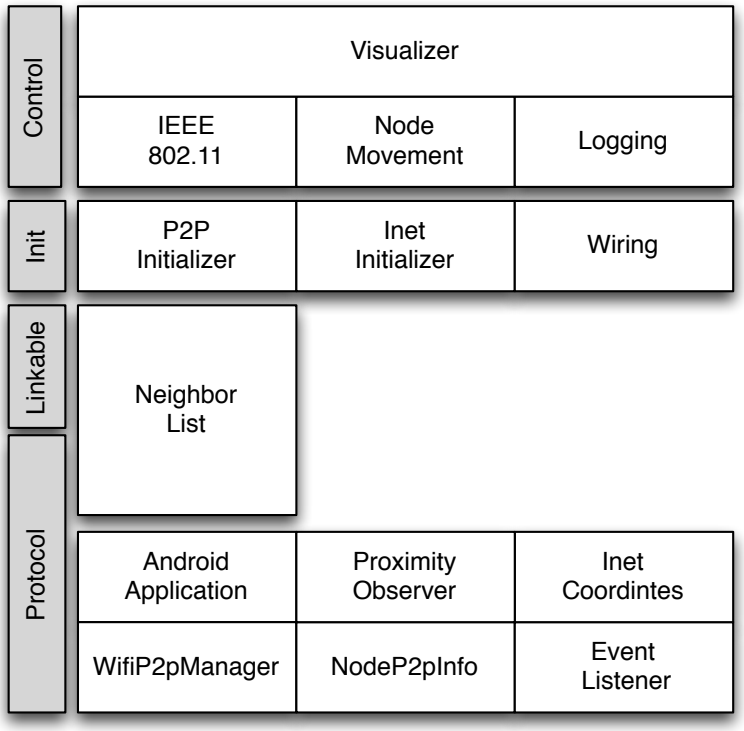


Figure 3.2: *Main components of the WiDiSi implementation*

An initial limitation is that WiDiSi is a single-threaded simulator; this means that all its internal components synchronize around PeerSim cycles. Android applications, on the other hand, can be multi-threaded. To solve this discrepancy, one can decide to use PeerSim in a hybrid mode, i.e., one in which both its cycle-driven and event-driven engines are used. In the following examples, we explain how to solve the multi-threaded problem. Our aim here is to do all the processes inside one thread that are synced around the PeerSim cycle. It is inconsequential that this technique cannot be applied to all multi-threaded applications. In the end, one may find out that converting his/her multi-threaded application to a single-threaded application is not feasible at all. Assume we have a $runnable$ in Android application like below:

```
delayHandler.postDelayed(new Runnable() {
  public void run() {
    if (condition){
      // the code
    }
```

```
  }
}, 2000;
```

In this code, a new thread would be created in parallel with the main thread, and after 2000 ms "the code" will be executed. In order to remove this parallel thread in PeerSim, we use flags and counters inside the $nextCycle()$ method in the main activity. For this reason, we define a condition for the $nextCycle()$ method as follows.

```
long counter = 0;
boolean flag = false;
public void nextCycle(Node node, int pid) {
    if(flag && counter >= (2000/CycleLength){
     // the code counter = 0;
    }else if(flag && counter < (2000/CycleLength)){
     counter++;
    }
    cycle++;
}
```

Now whenever we are interested in starting the runnable, we only need to make $flag = true$. This will make the counter count for $(2000/CycleLength)$ cycles that would be equal to 2000 ms and then execute the code. In this way we mimic the behaviour of a parallel thread.

A second limitation is that we only support Bonjour service discovery; we do not support UPnP or other service discovery mechanisms. A third limitation is that the channel that connects the application to the Wi-Fi P2P framework and channelListener are not available since they are not needed in our simulations. In Android an instance of a *Channel* is obtained by calling a specific initialize method [13]. As a result, the initialize method is also not needed in our Simulator. A fourth limitation regards Android's use of listeners for asynchronous method calls to the API. In Android's implementation responses from an application are dealt with through listener callbacks provided by the application itself. There are two kinds of listeners that are used. Some listeners are used to inform the application whether a call to the framework has been successful or not. We assume that all method calls are always successful and therefore, did not implement these listeners. Others are used to inform the application that the required information is ready to be picked up. These are important to us, and we support the following ones:

- WifiP2pManager.ConnectionInfoListener: This is the interface for when connection info is available.
- WifiP2pManager.DnsSdServiceResponseListener: This is the interface for when a Bonjour service discovery response is received.
- WifiP2pManager.DnsSdTxtRecordListener: This is the interface for when a Bonjour TXT record is available for a service.
- WifiP2pManager.GroupInfoListener: This is the interface for when group info is available.
- WifiP2pManager.PeerListListener: This is the interface for when peer lists are updated.

NodeP2pInfo is a class that keeps information about a Wi-Fi P2P node. This information could be general device info like the device's MAC address, its name, its

battery level or remaining memory, or its processing capabilities. It also keeps track of P2P device statuses, e.g., connected, busy, or invited.

Finally, class *Event Listener* is used to generate intents and deliver them to waiting for listeners. An Android application, which has already implemented the required listeners, will receive these intents whenever a Wi-Fi P2P related event occurs in the node or the node's proximity.

3.4.2 Network Dynamism and Proximity Manager

In order to make the network dynamic we need to be able to add and remove nodes from the network based on a user-defined pattern; we also need to move the nodes over time. The former is already provided by PeerSim. Indeed, PeerSim can add or remove nodes to the network at a specified period, or based on a given period. The latter, on the other hand, is not provided by PeerSim. To move nodes inside the network, and keep the network up to date, we defined and implemented the following concepts and classes:

- The geo-location of each node is based on X/Y coordinates; a node's location is kept within its instance of the *Inet Coordinates* Protocol.

- We provide a location to each node when they are added to the network. How this is achieved will depend on the scenario, and it is the responsibility of the *Inet-Initializer* component. This component is a PeerSim Initializer.

- Moving nodes inside the network is scenario dependent, and is achieved through the *Node Movement* Control component. Since *Node Movement* is defined as a PeerSim Control, it has access to all the Protocols and to the Linkable that are running inside each node. The Node Movement component exploits this to change the position of the nodes, as well as to update the proximity list of each node based on its new location and Wi-Fi Direct radio range. The proximity lists are maintained by the *Neighbor List* Linkable Protocol, which is installed on each node.

- *Proximity Observer* checks the *Neighbor List* Linkable/Protocol component for changes. If any change happens in the node's proximity, it is announced to the event listener for further actions.

3.4.3 Node Initializers

Whenever a new node is created, the PeerSim simulator proceeds to install all the available Protocols on it. However, these Protocols need to be initialized to have a correct state. The first thing that needs to be initialized after a node is created is its neighbor list. This is done through a *Wiring* class that initializes the *Neighbor List* Linkable/Protocol of the new node. It also updates the corresponding lists of these neighboring nodes, so that the information are correctly synchronized. After wiring has been performed, the *P2P Initializer* starts the *nodeP2pInfo* Protocol.

3.4.4 IEEE 802.11

The effects of the IEEE 802.11 MAC and PHY Layers and the wireless channel can be summarized as i) a package drop rate and ii) a delay for package transfer. These values are not fixed once and for all. Instead, in order to have an accurate model, we need to calculate these for each package transmission. These are essential calculations for our simulator and are performed by the *IEEE 802.11* Control component. The current calculations of drop rates and delays are quite basic. However, they have shown to be accurate enough to capture the behavior of a Wi-Fi Direct enabled device.

The drop rate may depend on many criteria, such as distance, congestion, and so on. The distance, however, is the most important one. By increasing the distance between two peers, the strength of the receiving signals will decrease. This will result in receiving the package with errors, which means an increase in package drop rate. The received signal strength in free space was calculated by H. T. Friis as $PathLoss(dB) = 20log_{10}(d) + 20log_{10}(f) + 32.44 - G_{tx} - G_{rx}$. *d* is the distance of the receiver from the transmitter (km), f is the signal frequency (MHz), *Gtx* is the gain of the transmitter antenna relative to an isotropic source (dBi), and *Grx* is the gain of the receiver antenna relative to an isotropic source (dBi). To check whether a package can be delivered, we follow the same technique used by NS-2 [79]. We use a threshold for signal strength at the receiver's end. If the receiving signal strength, calculated with the above formula, is below the threshold the package cannot be transferred. The delay, on the other hand, can be pre-calculated using another network simulator that supports IEEE 802.11, like NS-3 or Jemula802.

For the sake of greater flexibility, we also allow users of the simulator to apply their own calculation techniques through a plug-in mechanism.

3.4.5 Network Visualization and Logging

One of the main characteristics of a simulator is its ability to enable debugging. In this work, we have developed a PeerSim Control called *Logging* that can output debugging data to a text file.

We also provide the *Visualizer* component; it is a PeerSim Control that can help understand what is actually going on inside the network. We visualize the network in real-time using the Gephi Toolkit API [10]. Gephi is an open-source network analysis and visualization software package written in Java. The toolkit accepts a graph (directed or undirected) as an input and provides many useful tools to layout the graph, manipulate it by changing the size or color of its nodes, provide statistics, as well as filtering capabilities to select nodes and/or edges based on the network structure or available data. Figure 3.3 shows an example of the device-role visualization. In this figure, nodes are represented as circles and edges are used to display the connections between them. The size of the group owners accounts for the number of clients that they are currently serving. The color indicates the roles and the status of the nodes. In addition to visualizing the graph, other useful data are also made available in real time, such as the present and simulated time, the number of nodes within the network, the number of connected nodes, and so on.

Figure 3.4 shows a screen shot of MAGNET on the simulations environment. As mentioned earlier, different colors indicate different roles or status of devices inside the

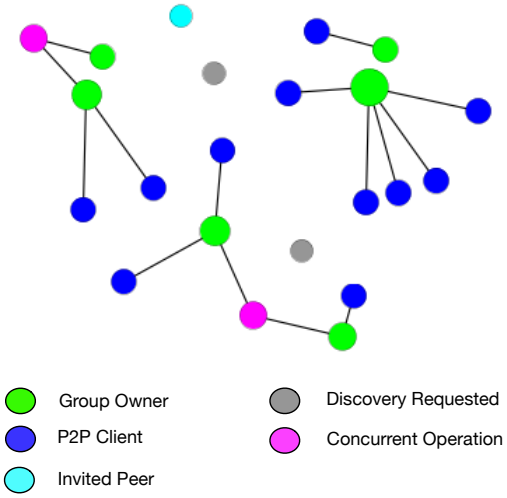


Figure 3.3: *Roles and status of the nodes*

network. The links indicate that a node belongs to the aforementioned group (either as a P2P client, colored blue, or a legacy client, colored pink). If a client is connected to two different groups, it means that it is playing the role of a bridge (red nodes). The API provided in WiDiSi would let the application developer fine tune the visualization parameters, such as screen refresh rate, quality of the images, and size and colors for different roles inside the Wi-Fi Direct network.

In addition to the main visualization windows, which help the user to watch the real-time changes in the topology of the network, there are two other windows that help the user to monitor the quantitative values, such as the number of messages that has been sent or the real-time logs. Figure 3.5(a) shows screen shots of the control/monitor window. The control/monitor window lets the user apply new network parameters like the number of nodes or the dimension of the simulated area at run time. Furthermore, it illustrates all the necessary real-time network data to help the user understand the status of the network better. This information includes the real-time and the time inside the simulator, current number of groups, number of connected devices, the current PeerSim cycle, the length of each PeerSim cycle, the average shortest path between all the nodes inside the network (in terms of number of hops), and so on. Moreover, the violation monitor section informs the user about any violations against the Wi-Fi Direct specification that have happened so far. The user can print out these violations to investigate further.

Figure 3.5(b) illustrates a screen shot of the logging window. The logging window prints out some other (less important) logs which may be useful for the user, such as the groups SSID and PassPhrase, The nodes that failed to join a group and so on. WiDiSi provides an interface mechanism to help the user to print only the logs that he/she is interested in and discard any other logs.

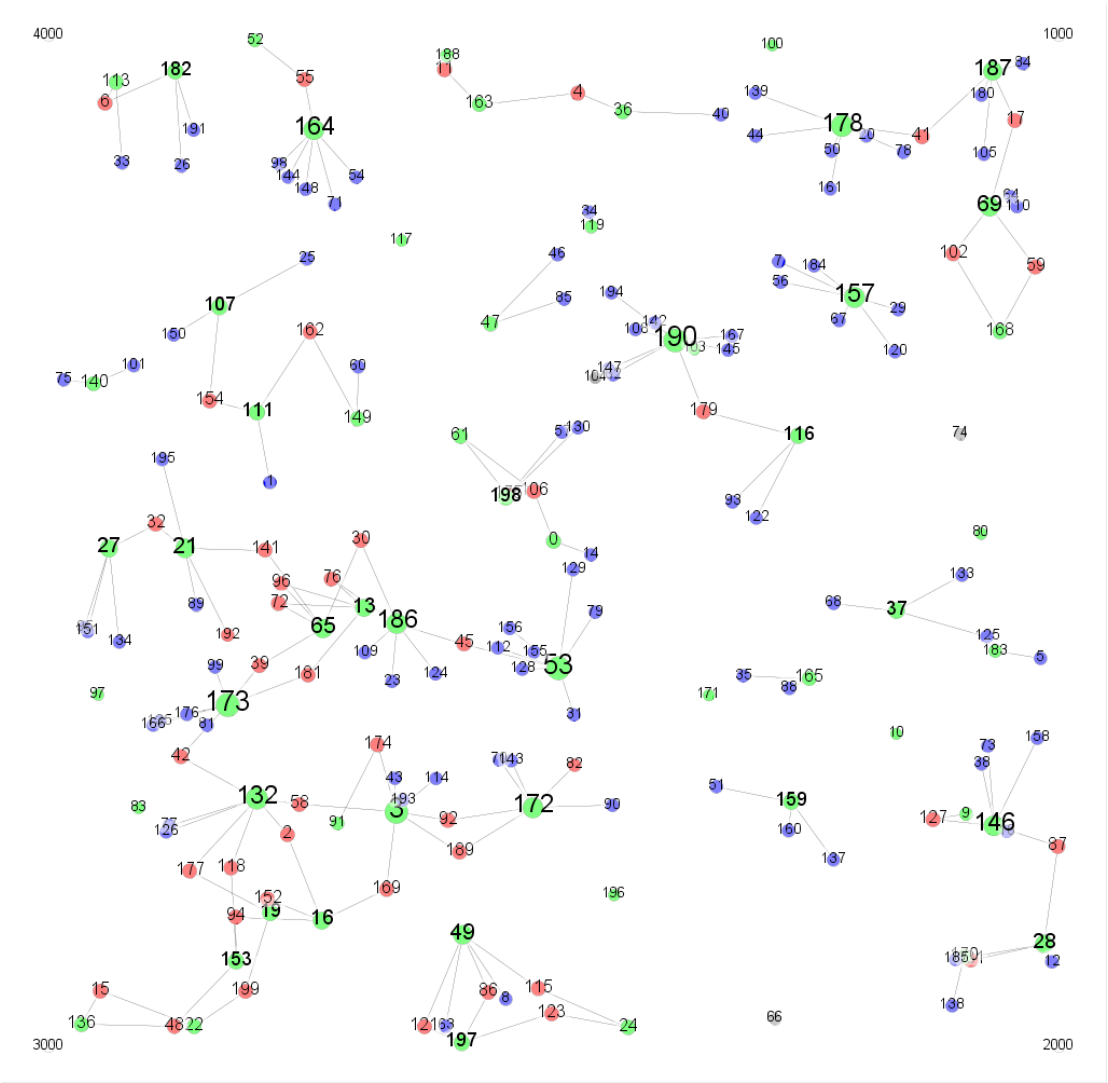
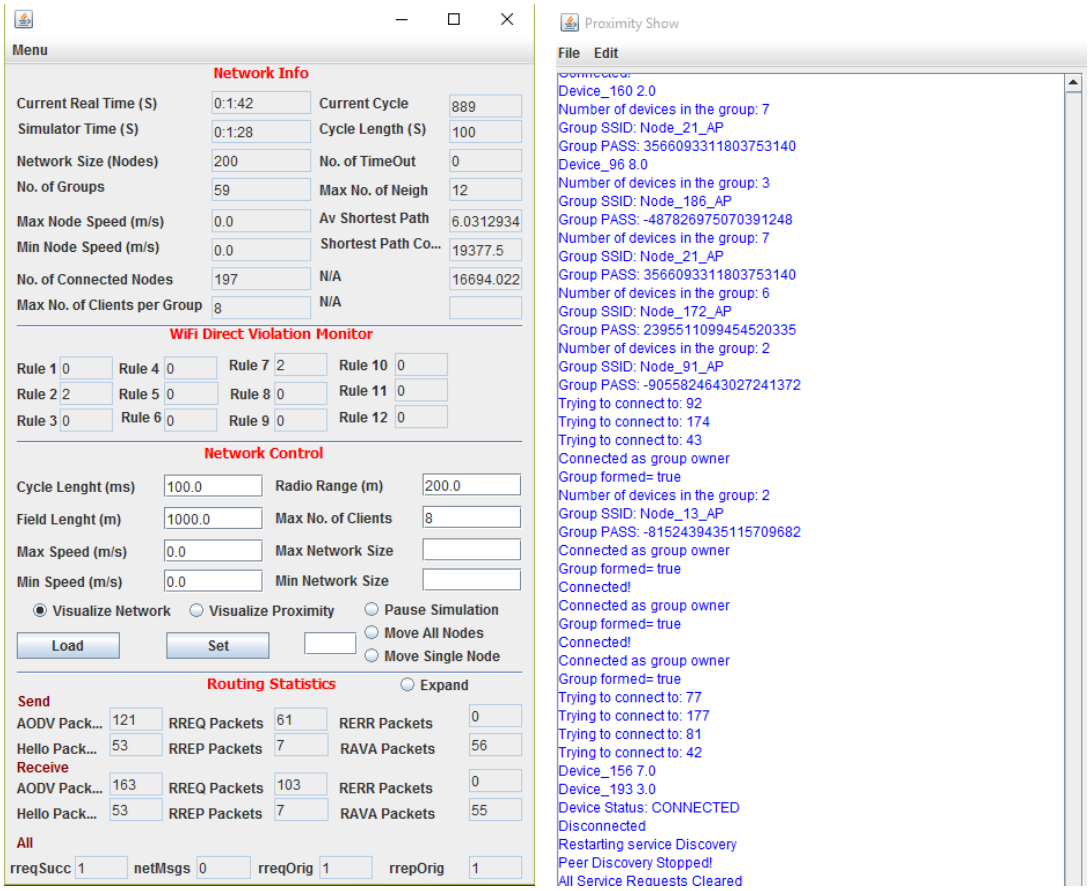


Figure 3.4: *A screen shot from the main visualization window*



(a) A screen shot of the control and monitor window

(b) A screen shot of the logging window

Figure 3.5: *Real time control, monitor and logging windows*

3.5 Evaluation

This section presents the tests we developed and the results we obtained while assessing WiDiSi.

The first experiment checked the behavior of WiDiSi using rules and monitors. The aim was to find out whether the simulated nodes behave like real Wi-Fi Direct devices in a Wi-Fi direct network. To this end, we defined over twenty rules that a Wi-Fi Direct device must comply with, as well as rules that state what the simulator must *not* do. For example, a general rule says that *A device cannot connect to another device outside its proximity range*, a more specific rule states that *A client cannot connect to another client directly*. These rules are further divided into three categories: general requirements that are forced by radio propagation in space, Wi-Fi Direct rules that are defined by Wi-Fi P2P specification, and OS limitations that are forced by Android. In the following, the most important ones have been as follows.

General rules

- A peer cannot connect to another peer outside its proximity range
- A group cannot consist of more than the pre-defined number of peers
- A peer cannot discover more than the pre-defined number of devices and services
- A peer cannot discover devices and services outside its radio range

Wi-Fi P2P specification rules

- A client cannot connect to another client directly
- A peer cannot discover other peers or services if they have not started peer discovery
- A group formation cannot take more than 15 seconds after a connection request has been received
- A peer cannot have access to group data if it left the group or if the group is not available anymore
- A group owner is always discoverable
- When two P2P devices negotiate to decide the GO role, the device with higher intention should become the group owner
- If the group owner fails, the group should be terminated
- The group owner role cannot be transferred inside a group before terminating the group
- Events and notifications should be dispatched as discussed in section 3.4.1.

Android rules

- A client cannot communicate with another client in the same group directly (the message should be passed at the MAC layer through the group owner - channel delay will be applied)
- The device/service discovery in a client remains active until a connection is initiated, or a P2P group is formed[4]

These rules cover the most important aspects of Wi-Fi Direct behavior. The application developer can guide the violation monitor component in WiDiSi to check all or part of these rules when needed.

We associated each rule with a corresponding monitor and checked the correctness of the simulator in two example scenarios. In both cases, the target application was a variant of a simple chat application over Wi-Fi Direct. In this application, devices chat with each other autonomously without any user intervention. We considered constant delays for the main delays discussed in Section 3.4. For example, we used Jemula802 to calculate an average for `Channel_Delay`, and discovered that it takes around 400 milliseconds to send one package, at the network layer, from a client to an access device. As for the other delays, literature provided the information we needed. For instance, Levy [19] suggests a value of 200ms for the average time taken by Android to carry out a simple action, and we used it as our `Internal_Processing_Delay`. Note that the user can configure all these parametric values to make the simulation environment more suitable to his/her needs.

The first scenario we considered was a dynamic situation with few devices. We simulated a one square kilometer area with an average of 200 Wi-Fi Direct devices. On top of that 100 devices were allowed to enter or leave the area periodically. Devices moved around with a speed between 0 to 20 m/s. We ran the tests for sixteen different speeds with identical simulation parameters. To keep the parameters constant for the entire evaluation campaign, we used the same random seed numbers for every random value, such as position and direction of movement.

The second scenario referred to a larger-scale, yet less dynamic, situation. We simulated a one square kilometer area with 10,000 devices. In this case, devices could move around, and their number was fixed. As for the first scenario, the application continuously searched for peers and attempted to connect to those found in range. Since nodes were moving in arbitrary directions, they would connect and disconnect. When two or more nodes were connected to each other inside a group, they exchanged their values and adopted the value that was closer to the calculated mean. The standard deviation of all the values inside the nodes of the network showed how fast the values inside the devices could converge, and therefore how fast these nodes could connect and exchange data. Figure 3.6 represents the (virtual) time needed to reach a particular standard deviation in the network with various speed limits. The horizontal axis represents the speed of the different nodes, and the vertical axis represents the time needed for the standard deviation to become less than 0.15.

Figure 3.6 indicates that when devices move very slowly, the convergence is slow. However, when the speed increases the time needed to reach the threshold decreases.

[4]This rule is not fixed in different Android version. The application developer should consult the Android API documentation for the correct implementation

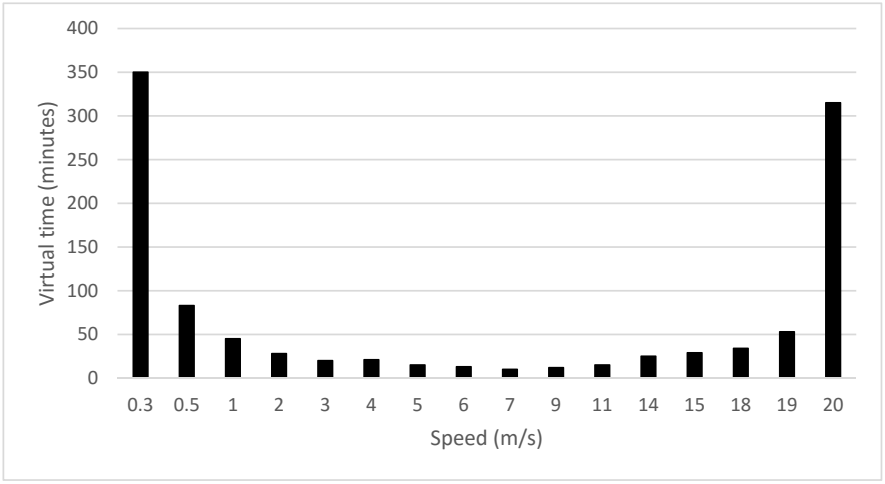


Figure 3.6: *Time needed for the standard deviation to become less than* 0.15 *(Scenario 1)*

When nodes move faster, they can discover other nodes and connect to them more frequently, and this means that less time is needed to reach the target deviation point. When the speed becomes 9m/s, the time starts increasing again. The trivial reason behind this is that when nodes move too fast, they do not have enough time to complete the procedure required to form a group: they may start negotiating for group formation, but then leave the proximity range before completing it. The only lucky case is when two nodes move in the same direction.

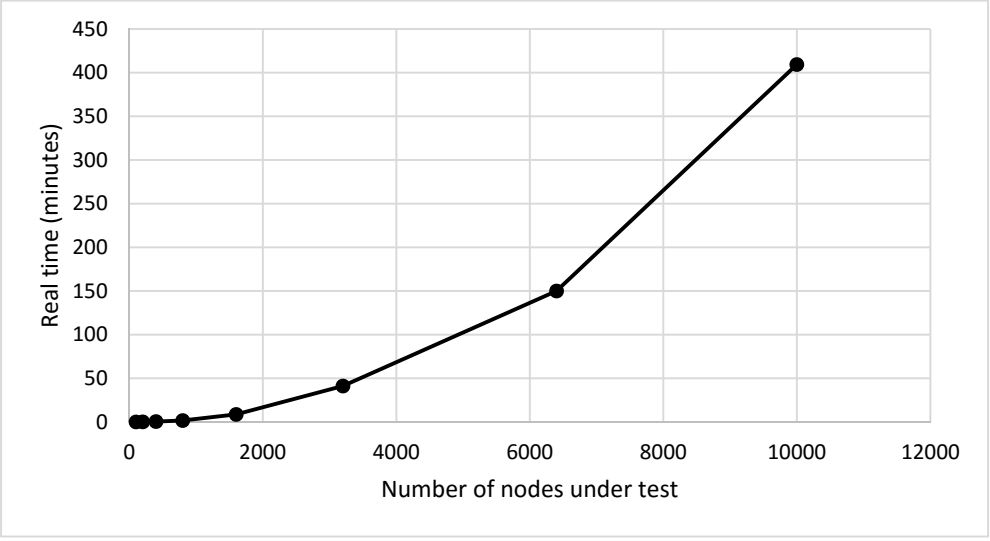


Figure 3.7: *Time needed for a* 10*-minute simulation with different numbers of nodes (Scenario 2). We assume, a second corresponds to 10 PeerSim cycles*

Scenario number two tested the time required to simulate a large-scale scenario. Figure 3.7 illustrates the time needed for a 10 (virtual) minute simulation. The horizontal axis represents the number of devices involved; the vertical axis represents the time needed to simulate 10 minutes in the virtual world. (Network visualization was disabled to obtain the maximum response time.) This test was performed on an Intel®Core™i7-

2670QM CPU 2.20 GHz with 6 GB of RAM and 64-bit Windows operating system. The time increases exponentially with the number of nodes. With 2000 nodes the simulator time is almost identical to wall-clock time. Needless to say, a faster computer can help with high numbers of nodes.

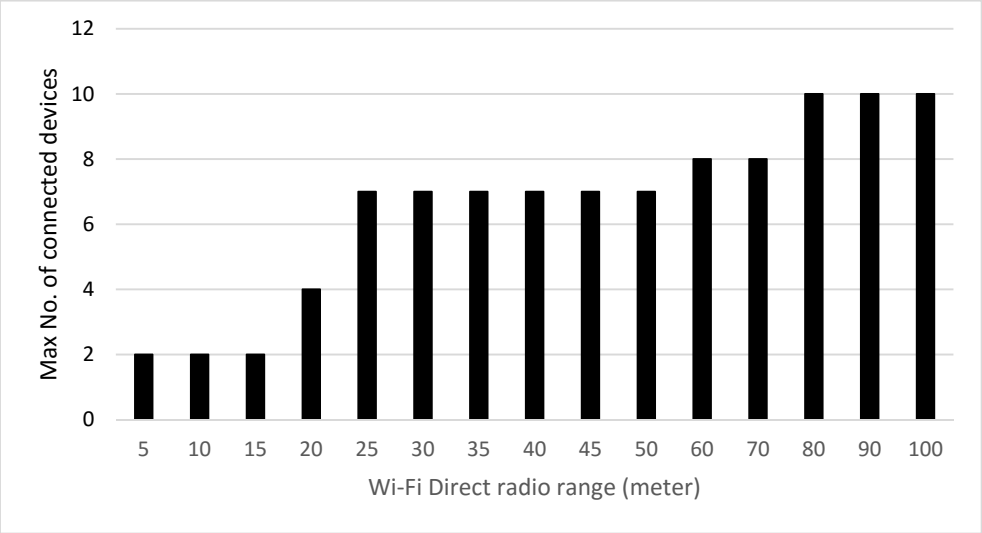


Figure 3.8: *Maximum number of possible simultaneously connected devices for various radio ranges*

The final experiment shows the correlation of connected devices and wireless radio ranges (see Figure 3.8). The test was performed on ten devices in a 50 by 50 meter area. The horizontal axis represents the radio range of each device in meters; the vertical axis represents the number of connected devices. As we can see, increasing the radio range resulted in higher numbers of discoverable devices and higher choices for connection. However, the number of connected devices remained below the maximum number of possible connections, due to higher package loss on longer distances.

CHAPTER 4

Proposed Middleware Infrastructure

This chapter presents *MAGNET*, a novel middleware infrastructure that exploits Wi-Fi Direct to provide a reliable and stable communication means for large numbers of mobile devices. This self-organizing middleware abstracts the multi-hop communication process by autonomously maintaining connectivity among devices. *MAGNET* also provides a discovery mechanism that exploits the MAC address of the different devices or the services they offer. We tested *MAGNET* on both WiDiSi and real Android devices. The evaluation results illustrate the effectiveness of the proposed middleware in discovering and maintaining the connectivity in large-scale dynamic scenarios. The assessment of magnet has been discussed in details in Chapter 5.

The following chapter is organized in to two sections. Section 4.1 presents the architecture of *MAGNET*, while Section 4.2 describes its implementation.

4.1 Architecture

MAGNET, our Wi-Fi Direct-based middleware, supports reliable, multi-hop communication among large numbers of Android devices. To better frame the problems MAGNET solves let us introduce a simple real scenario: the off-loading of LTE traffic in a crowded district of a city. In this scenario the mobile operator must manage a large number of devices that enter and leave each other's proximity at will. Our proposal is to use a solid and robust Wi-Fi Direct network among the proximal devices to reduce the traffic the LTE tower needs to manage.

As already explained, the first step towards the creation of a Wi-Fi Direct network is the creation of groups. Since each group can only accommodate a few devices, we need to create multiple groups and for each define a group owner; this must be done in a homogeneously distributed way. Owners are selected taking into account processing

capabilities, battery life, and their intention to move. Note that group owners must carry out both the "usual" functionality required by Wi-Fi Direct (e.g., act as DHCP servers) as well as what is required by MAGNET, i.e., it must oversee routing and decide on interconnecting groups.

After creating a first set of groups, which are established freely, some devices (nodes) may remain unconnected. Once again, this may be due to group saturation, congestion, internal errors, or delays in connection requests. MAGNET oversees the creation of groups, and manages the nodes that remain unconnected allowing them to become part of a group. If all requests to join the proximal groups fail, the unconnected device creates an autonomous group and becomes a zero-client group owner. As soon as each device is part of a group, MAGNET starts connecting groups to form an overlay network that allows all groups, and the devices therein, to be connected. Finally, MAGNET provides multi-hop routing among these devices to enable unicast and multicast communication. The layered view of Fig. 4.1 illustrates the functionality embedded in MAGNET and exemplifies the components installed on each Android device.

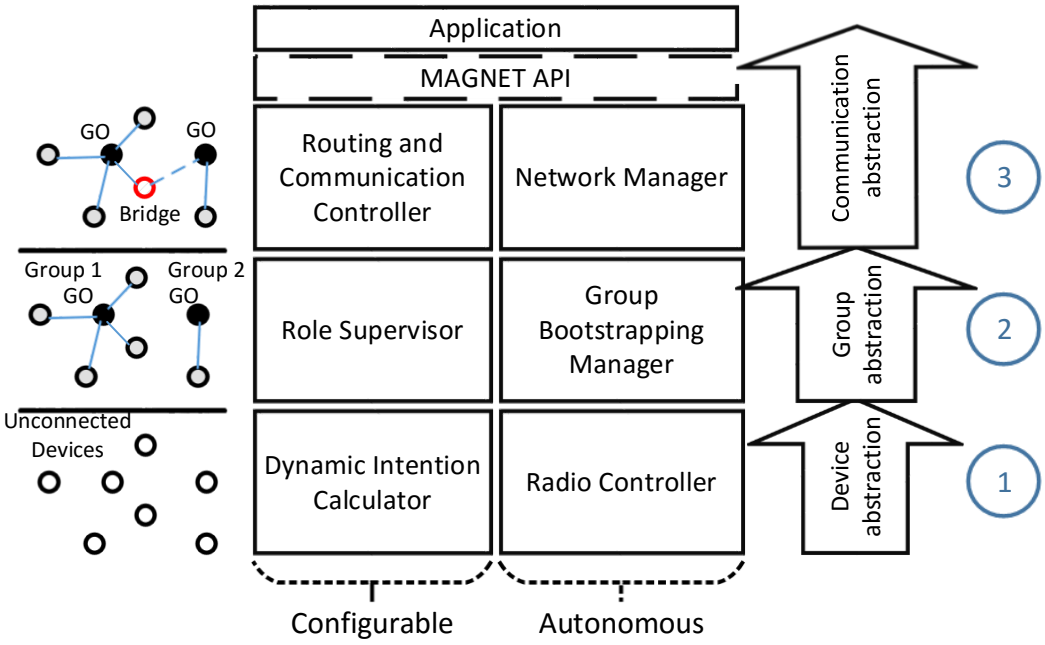


Figure 4.1: *MAGNET architecture*

The architecture consists of three main abstraction layers. Each layer comprises two main components: a **user-configurable** part and an **autonomous** part. Most of the functionality provided by MAGNET is carried out autonomously to keep a Wi-Fi Direct network connected without any user intervention. Nevertheless, some user preferences and configurations must be taken into account.

4.1.1 Device Abstraction

The first layer is a *platform-dependent* layer that abstracts the device. This layer consists of two main components. Component **Radio Controller** utilizes the Wi-Fi and Wi-Fi Direct APIs provided by Android to perform discovery and communication. This component is responsible for providing UDP/TCP socket communication between group members. Moreover, it is responsible for registering and advertising the services provided by a device, amongst which a special-purpose service called *MAGNET*. This service contains metadata including the intention of the device to become a group owner. It is also responsible for discovering other proximal MAGNET-enabled devices by exploiting the Wi-Fi P2P low-level service discovery mechanism[1].

Component **Dynamic Intention Calculator** is in charge of computing the value of the aforementioned intention, and providing it to the radio controller when needed. The computation takes into account the current status of the device, i.e., its remaining battery, available memory, and computing power. The user can configure the weights of these criteria.

The other layers of the middleware are platform-independent.

4.1.2 Group Abstraction

The second layer is in charge of ensuring that the device is part of a Wi-Fi Direct group in any circumstance. For instance, if the device moves close to a group, moves away from it, or must face saturation, this layer works to keep the node connected.

While devices are busy forming groups, we assume they define their roles (owner or client) individually by looking at the intention values of their proximal devices. Each device waits for a specific amount of time to be able to discover all the other devices and their intention values. If the intention value of a device is higher than all the other discovered values, the device autonomously becomes the group owner and invites the other nearby nodes to join the group. It also continues to accept connection requests, up to the maximum capacity. If the intention value of the device is not the highest among the sensed ones, it simply waits for a given amount of time to see if another device (with a higher intention value) becomes the owner. The time each node will wait is configurable on each device. Short delays could mean quicker group formation, but also less optimal grouping since decisions are less meditated and there is no time to wait for a better, more complete configuration. Note that, given our overall goal, the best group formation is when the devices with the highest intention values become owners, and all groups reach the maximum number of clients. The decision on the role of a device is the responsibility of a component called **Role Supervisor**.

MAGNET's owner election is borrowed from leader election algorithms [43,62,71]. Specifically, Raychoudhury et al. [71] propose a k-leader election algorithm in which k leaders are elected, in a distributed setting, based on their *weights*. Diffusing computation [40] is adopted to collect the information about weights for electing the leaders. MAGNET elects leaders (owners) locally, by only looking at the proximal devices, while the algorithm in [71] takes a global approach and considers all the elements in the system.

[1]Service discovery is needed to discover MAGNET-enabled devices. Device discovery, in contrast, would search for any proximal device.

Component **Group Bootstrapping Manager** is responsible for the actual creation of groups. It is also in charge of managing connection requests, failures, new arrivals, saturations, and empty groups (owners only). MAGNET handles all these events autonomously. For instance, if a device moves out of the proximity of its group, it gets disconnected and should perform the group-bootstrapping procedure again to connect and become a member of another group. Connections and disconnections are kept hidden to the user (application level). As soon as a group exists, group bootstrapping enables the communication between devices inside a group based on MAC addresses[2].

4.1.3 Communication Abstraction

The third layer is in charge of inter-connecting Wi-Fi Direct groups and providing multi-hop connectivity between devices. The main goal of this layer is to ensure that a group is connected to as many groups as possible in any circumstance. Since Android's Wi-Fi Direct does not allow a single device to be both the owner of a group and a client of another one, we cannot connect groups through Wi-Fi Direct, but we must exploit other "legacy" protocols like standard Wi-Fi or Bluetooth. Since a Wi-Fi Direct group can also host legacy Wi-Fi clients, which see the owner as a standard access point, MAGNET exploits Wi-Fi connectivity, which results in better integration compared to using Bluetooth.

Each group owner decides which of its clients must connect to which proximal groups. These clients, which act as *bridges*, then use conventional Wi-Fi APIs to try to connect to the other groups. (The Wi-Fi Direct interface is already busy with the participation in the first group.) If a device plays the role of group owner, component **Network Manager** is in charge of discovering and managing both failing bridges and new proximal groups. Zero-client groups can connect to the other groups via bridges, but they also send periodic connection requests to the proximal groups, even to those that have already rejected their requests. If the device is a client, it must only inform its owner about any change in the proximity, that is, it must inform the owner about the appearance of new groups or nodes.

The network manager of a group owner is responsible for making local decisions, by only considering nearby groups. The group owner must connect its group to as many proximal groups as possible, and it must also avoid creating multiple bridges with the same group. In the extreme case in which all groups are in the proximity of one another, this strategy would tend to create a fully connected mesh network. Although creating a fully connected mesh would result in higher message costs[3] and more congestion, we prefer to keep this option for two reasons: (a) redundancy helps keep the stability of the network in dynamic environments, and (b) there is an implicit upper bound to the number of inter-group connections. This limitation is due to the maximum number of clients each group can host. For instance, if the maximum number of clients per group were four, it would not be possible to create a fully connected mesh network with more than five proximal groups. However, a fully connected mesh network will only happen in the extreme case in which all groups are in the proximity of one another.

Component **Routing and Communication Controller** is installed onto each group

[2] In Android, the communication within a Wi-Fi Direct group is based on IP addresses.

[3] Message cost here refers to the number of messages that should be propagated to perform a task such as finding the shortest path to destination or recovering a broken link.

owner, and is in charge of routing and communication between groups. Routing can be based on either unique IDs (e.g., MAC addresses) or well-known service names (i.e., a name that is known to every MAGNET-enabled device). In both cases, the source node initiates a discovery to find whether the destination device/service is available within the network. The dynamism of mobile devices requires that this discovery be repeated several times (to increase the chances of success) within a time interval that is set by the user.

When a device receives a message, and it is not able to serve the request, i.e., its id does not match the requested one or it does not offer the requested service requested, the device simply forwards the message to the next hop. If, on the other hand, the requested id matches the device's id, the device accepts the message and generates a reply with the best path to reach it. Similarly, if the device offers the same service as the requested one, it accepts the message, generates a reply, and forwards the message to the next hop. If multiple instances of the "same" service are retrieved, the application is in charge of deciding which one to use. To avoid live-locks devices discard the messages they have already forwarded. As for identifying the next hop, since MAGNET generates an ad-hoc network, any routing algorithm for mobile ad-hoc networks would be suitable. The current version of MAGNET supports two different solutions, to better support diverse application domains. The first is the widely-used Ad-hoc On Demand Distance Vector (AODV) routing algorithm [65]. Although AODV is designed for dynamic scenarios, the message cost associated with its route maintenance solution increases with the mobility of devices [64]. Therefore, for highly dynamic application domains, we propose a simplified version of AODV without any route maintenance protocol. This is described in more detail in Section 4.2.

As already said, group owners wait for a given amount of time before making decisions about interconnecting their groups. These delays allow for a better formation of the network; faster decisions may decrease the number of connected groups because of the limited number of answers. After an amount of time that is set by the user, the group owner takes its decisions even if some clients have not finished sending their results. There are also other configurable parameters related to routing that the user can set, to better support the peculiarities of the different scenarios and to end up with smaller message costs in routing (Section 4.2).

4.2 Implementation

MAGNET is implemented in Java, the most common language for implementing Android applications. MAGNET's implementation had to cope with the intrinsic limitations of Wi-Fi and Wi-Fi Direct, and of their implementation in Android. Android classes *WifiP2pManager* and *WifiManager* provide the functionality for exploiting these two protocols. This section explains how we have used them to implement MAGNET.

As soon as an unconnected device discovers another MAGNET-enabled device in its proximity, it starts bootstrapping a group, i.e., it starts defining roles and dispatching connection requests. However, in Android each connection request requires user approval, a severe limitation for the operation of the middleware. In order to overcome the problem, MAGNET only supports user approvals through the so-called Push Button

Configuration (PBC), and disregards the Pin Method Configuration (PMC), and uses a mechanism introduced by Octoblu[4] to intercept the confirmation dialog box and accept the connection autonomously.

Whenever a client discovers a new group, it informs its owner so that it can decide about inter-group routing. First, each client sends its group owner the list of the (other) groups it can "see" and their distance, estimated through the Received Signal Strength Indicator (RSSI) provided by the wireless interface. The owner then decides the bridges that must be set, i.e., the pairs $(client, group)$ that need to be established. Connections with the highest quality are preferred, i.e., connections that have the smallest distance between devices. The number of connections that must be set is always the minimum between the number of clients in the group and the number of groups nearby. A group cannot create more connections than the number of its clients, and it can only create one bridge with another group. To find the best set of bridges, MAGNET uses a Depth-First Search (DFS) algorithm[5].

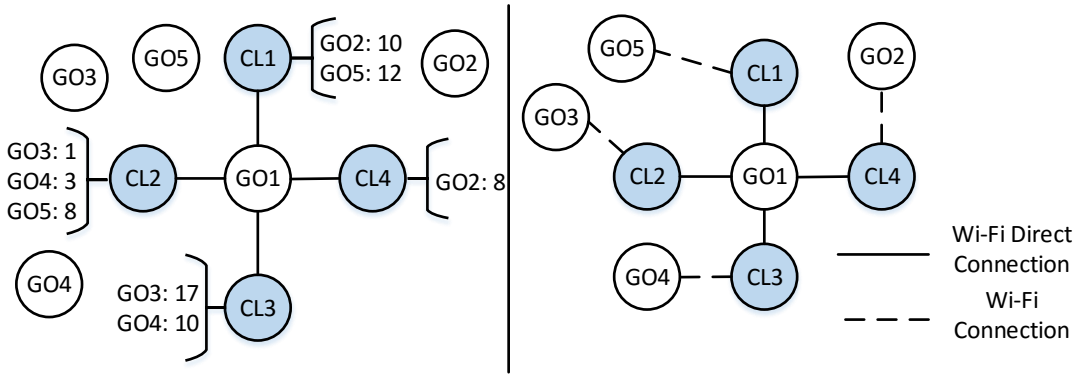


Figure 4.2: *Example group topology and bridges*

Fig. 4.2 illustrates a scenario that comprises a group with 4 clients, and 4 other groups represented by their owners. This means that GO_1 can create up to 4 bridges. If we were to assume that every client ($CL_{1..4}$) can see all the groups ($GO_{2..5}$), we would end up having the maximum number of possible connections. Instead, the most common case would be to have clients with limited visibility of one another; in this case we would attempt to maximize the number of connections using the DSF algorithm. Given the situation of Fig. 4.2, the algorithm searches for combinations that lead to 4 connections, i.e., the maximum in this example. Again, the algorithm would select: $\{(CL_1, GO_5), (CL_2, GO_3), (CL_3, GO_4), (CL_4, GO_2)\}$ and $\{(CL_1, GO_5), (CL_2, GO_4),$-$(CL_3, GO_3), (CL_4, GO_2)\}$. The former would be better due to the reduced distances between nodes, higher signal strength, and better throughput. The final selection is obtained by exploiting the Weapon-Target Assignment (WTA) problem [49], a class of combinatorial optimization problems. The original formulation tries to find an optimal assignment between a set of weapons (clients) of various types and a set of targets (groups) to maximize the total expected damage (throughput) to the opponent. In our case the optimization problem can be stated as:

[4] `https://github.com/octoblu/alljoyn/blob/master/alljoyn/alljoyn_java/helper/org/alljoyn/bus/p2p/WifiDirectAutoAccept.java`

[5] Since the search space is small, any greedy algorithm such as DFS or Breadth-First Search (BFS) would be fine. Note that given k clients and n groups, the number of all possible combinations is equal to $(n)_k = n!/(n-k)!$.

$$
\begin{aligned}
min\ J &= \sum_{i=1}^{N} V_i \prod_{j=1}^{M} (1 - p_{ij})^{x_{ij}} \\
\text{subject to: } & \sum_{i=1}^{N} x_{ij} = 1, \quad j = 1, 2, ..., M.
\end{aligned}
\tag{4.1}
$$

where N is the number of groups, M the number of clients, V_i is the value associated with group i (the default is 1, but groups could be ranked according to their size, and service advertisement used to inform proximal groups about these values), and P_{ij} is the probability that weapon j kills target i, which in our case becomes the quality of the channel between the two entities. The solution then can be represented as:

$$
x_{ij} = \begin{cases} 1 & : \text{if client } j \text{ assigned to group } i \\ 0 & : \text{otherwise} \end{cases}
$$

The objective is to maximize the throughput, while the constraint is that each client can only connect to one external group. This solution provides the best combination; moreover, the user can always change parameters v_i and P_{ij} and guide the selection.

After finding the best set of bridges, the owner sends a command to its clients to ask them to connect to the different groups via their regular Wi-Fi interface. However, the Wi-Fi P2P specification says that the group owner should act as a "standard" Wi-Fi access point and have an SSID and a Passphrase[6]. This means that a client must search for proximal groups using the regular Wi-Fi scan procedure, and that groups must reveal their Passphrases through the Wi-Fi Direct service advertisement/discovery. As soon as a device becomes a group owner, it registers a service called **GroupOwnerPassPhrase**; its metadata contains the SSID and Passphrase of the group owner. A client that would like to connect to a group should then discover this service. We used the same technique to advertise and discover the TCP/UDP Socket port number when the default port number (4545) is already occupied on a device by another application.

As already said, we can easily foresee two different application domains where MAGNET can help: one with limited mobility and heavy payloads (e.g., video streaming at a stadium) and one with light payloads and extreme mobility (e.g., a chat service in a shopping mall). The routing in these two example situations must be managed carefully and different solutions can be taken into account.

For the first case application domanin we utilized a well-known routing protocol for wireless sensor networks called Ad hoc On-Demand Distance Vector (AODV) routing. This protocol was designed for mobile nodes to determine unicast routes to destinations within an ad-hoc networks [65]. It offers quick adaptation to dynamic link conditions, low processing and memory overhead, and low network utilization. AODV uses message broadcasting to find the shortest path to a destination, and uses diverse routing maintenance mechanisms to keep a route alive. However, route maintenance comes with a high message cost in very dynamic networks. The user can play with timeouts and route maintenance intervals to decrease the number of messages or decrease the broken route detection time.

For the second kind of application domain, that is in which the size of messages is small (and only occupies one UDP packet) and a fixed link between source and desti-

[6]We observed that SSID and Passphrase are not constant on the same device and may change in each group formation.

nation nodes is not necessary (like sending a notification or searching for a printer in the network), we propose an optimized heuristic routing protocol in which each message finds its way to its destination individually. Each device processes the incoming messages based on its role in the network (Fig. 4.3). Group owners are responsible for checking if the destination is within their groups; if it is not they forward the message to the nearby groups using the bridges. The routing is based on either the target MAC address or on a service name. If the routing relies on the MAC address, the message must be forwarded until it reaches the destination; if the routing relies on a service name, any device that offers a service with that name accepts the message and responds to it. The device that issues the request may then receive different responses from devices that support the requested service. To ensure loop-free routing, we use the same solution as AODV, i.e., we keep a history of forwarded messages for a given amount of time, and avoid forwarding already-sent messages. Since there is no route maintenance protocol in our solution, and only group owners and bridges act as message routers, the message cost is much lower than the one implied by the AODV.

No matter the routing algorithm, the main challenge for exchanging messages between proximal groups in a Wi-Fi Direct network is the IP conflict problem. Inter-group communication in Android has the following limitations:

- Android assigns a fixed IP address (192.168.49.1) to every group owner. Therefore, it is impossible for a bridge to create two TCP sockets to two group owners simultaneously.
- Android follows a weak-end system model; decisions about routing are only based on the destination IP address and service type [45]. Thus, when a device is connected via both Wi-Fi and Wi-Fi P2P, every unicast connection is automatically re-directed to the Wi-Fi interface since Android prioritizes Wi-Fi over Wi-Fi Direct [45]. This means that Wi-Fi Direct clients cannot use unicast communication.
- Group owners assign IP addresses to their clients individually without considering the other groups in proximity. Therefore, in a multi-group network, many clients may have the same IP addresses.

A few works have already tried to overcome these limitations. Funai et al. [45] propose a time sharing mechanism in which the bridge switches between two or more groups. Switching implies disconnecting from the current group, scanning for active nodes, and requesting to connect to a new group. Frequent connections, scans, and disconnections are time-consuming, and thus this solution is not feasible in many cases. Moreover, as discussed later in Section 4.1, connection and scanning requests are not always successful in Android and human intervention would be mandatory. Casetti et al. [34] suggest to make group owners act as legacy clients in other groups and then select a client to relay the packets at the application layer (Fig. 4.4(a)). To prevent IP conflicts, they utilize UDP broadcast (instead of TCP unicast) for the communication between group owners and their clients. However, this solution decreases the number of connected groups. Since a group owner can only be a client in one other group, the number of external groups a group can connect to are then restricted to one.

Similar to the solution proposed in [34], we also use UDP broadcasting between P2P clients and group owners to prevent IP conflicts. However, instead of making a

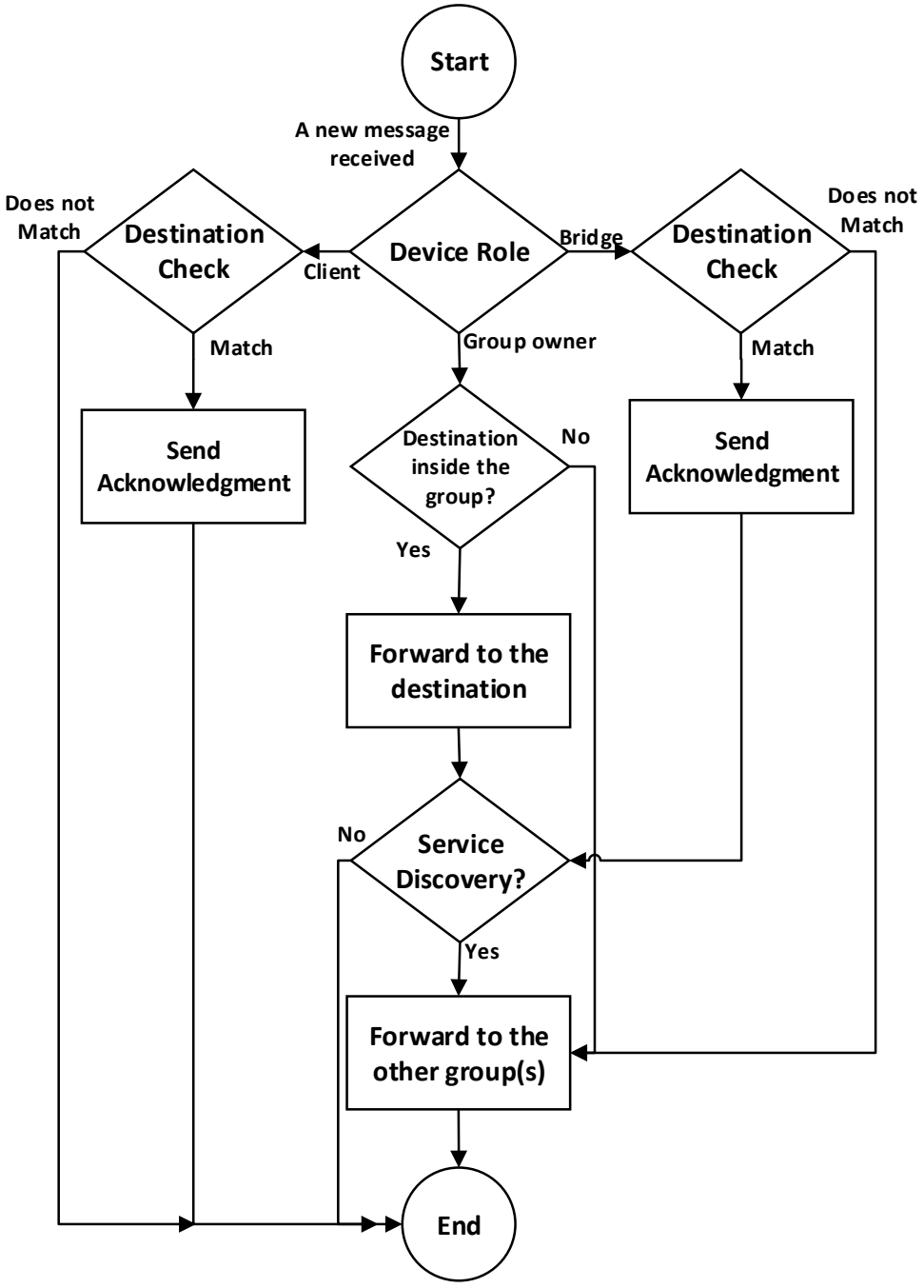


Figure 4.3: *Our routing algorithm for MAGNET*

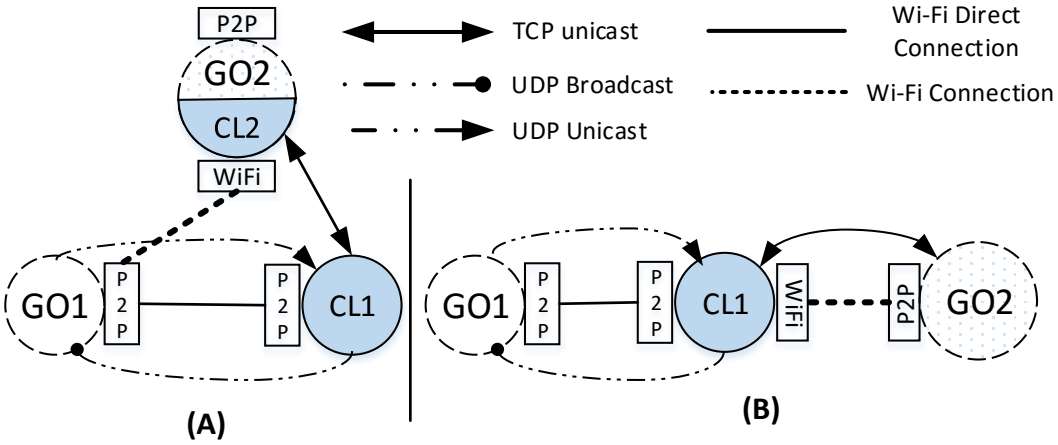


Figure 4.4: *Inter-Group communications with different strategies - (a) Group owner becomes a legacy client in another group and (b) A P2P client becomes a legacy client in another group (our solution)*

group owner become a legacy client in another group, we make one P2P client become a legacy client in another group and act as bridge between the two groups using the Wi-Fi and Wi-Fi Direct interfaces simultaneously (Fig. 4.4(b)). This way, the number of possible connections to external groups can be up to the number of clients in the group. For the bi-directional communication between these clients and the other groups over Wi-Fi, we can still use reliable TCP socket connections (as illustrated in Fig. 4.4(b)). The main drawback of this solution is that UDP does not implement any retransmission method, and it can be subject to data loss, which could be up to some 7% of the total packets transmitted [45]. However, some proposals already exist to provide reliable communication over UDP [46].

CHAPTER 5

Evaluation

In this chapter, we address research question four [**Q.4**] in section 1.2. This research question is related to the evaluation of the correctness and effectiveness of the proposed solution. We describe the validation of MAGNET in five different case studies and scenarios which have been introduced in section 1.2. The selected case studies are chosen from different application domains. Each case study address some qualitative and/or quantitative evaluation metrics. These metrics are defined in Table 5.1 and Table 5.2.

Table 5.1: *Qualitative Evaluation*

Measure	Assessment
Dynamism	Does the network self-organize in case of devices entering/leaving/failing?
Scalability	Is the middleware able to connect large numbers of devices? Does the network able to manage group saturation?

Table 5.2: *Quantitative Evaluation*

Measure	Metrics
Efficiency	Group and network bootstrapping time, delays
Routing message cost	the average number of routing messages to deliver one message
Delivery rate	the average number of delivered messages
Group Connectivity	the average number of devices that are connected to a group
Network Connectivity	the average number of devices each device has access to

To conduct a first assessment of MAGNET, we have considered a simple chat application, where every device tries to send "hello" messages to all the other devices sequentially. The size of sent messages is limited to embed them in single UDP packets. We used this application to conduct two different sets of experiments: one on physical devices and one through a simulator. We start this chapter by performing qualitative evaluation using the simulation environment. Later, we will extend these tests on the real devices as well.

5.1 Qualitative Evaluation

The qualitative evaluation can be organized into two broad categories: 1)dynamism, and 2) scalability. Each of these categories is divided further into several subcategories. This section describes the experiments we conducted to evaluate the effectiveness of MAGNET in dynamic and/or large scale scenarios.

5.1.1 Case Study One: A Ceremony

This case study aims to verify MAGNET behavior when facing a large number of devices. We consider a dense funeral with 500 people each with one smartphone(Fig. 5.1). The area is 100 square meters, and the radio range is set to 40 meter[1]. All devices in this scenario are static, and we only interested to observe the correctness of MAGNET when facing a large number of devices. Figure 5.2 shows the simulated version of the mentioned case study in the WiDiSi. As can be seen in Fig. 5.3 MAGNET can connect 499 devices out of 500 using 90 groups (*group connectivity*). These groups are connected to each other via bridges (red nodes). We observed that MAGNET was able to connect the majority of the devices to each other. Later in section 5.2.3 we provide some quantitative metrics in worst case scenarios to measure the level of the connectivity of the network.

5.1.2 Case Study Two: A Shopping Mall

The aim of this experiment is to validate MAGNET performance in a dynamic environment. The quantitative validation of MAGNET in a dynamic environment was performed in section 5.2. In this experiment, we consider a shopping mall (Fig. 5.4). The important characteristic of a shopping mall is the high mobility of people who hold the devices. In order to simulate such an environment, we considered a 500 square meter' district of a city with 200 mobile devices. We assumed that the Wi-Fi range is 100 meters [2]. The aim of this experiment is to evaluate MAGNET in a dynamic environment and test if the middleware is able to connect/reconnect the network when the following events happen:

1. A P2P client leaves the proximity of its group and enters the proximity of another group

[1] Although this experiment was held outdoors, high congestion would have caused the effective radio range to be minimal.

[2] The theoretical radio range of Wi-Fi in open space is 200 meters. However, we tried to be a little conservative, and, as a result, we chose half of the real range in this experiment.

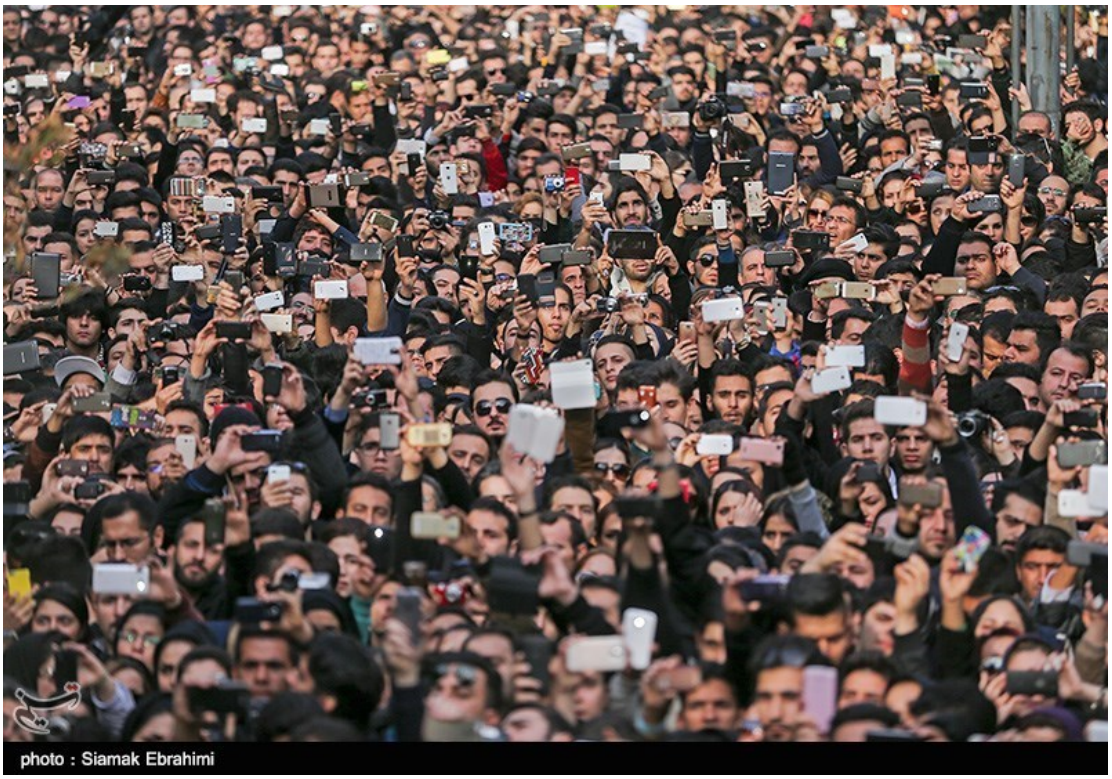


Figure 5.1: *A funeral in Iran*

2. A group owner leaves the proximity of its group and enters the proximity of another group
3. A bridge leaves the proximity of its groups, causing two groups to lose their connection
4. Two groups get close to each other
5. A new device faces a saturated group

We verified the correctness of MAGNET behavior by manually moving specific nodes to create the above mentioned events. We repeated this procedure 5x20[3] times. In each round, a random node was selected and was moved to observe the behavior of MAGNET. We observed that MAGNET was able to connect/reconnect the network in all tests without any errors. For the sake of clarity, we only illustrate one example (out of 20) for each of the five categories that are mentioned above.

Event.1 and Event.4 The first event is the situation where a P2P client leaves the proximity of its group. The expected behavior of MAGNET in such a situation is to reconnect the node to the network. Fig. 5.4 illustrates the simulated version of a shopping Mall and Figure 5.5 magnifies a part of the network. The node $n.121$, which has been identified by a green arrow, is manually moved to leave the proximity of its group. Since there is not other group in the proximity of the node $n.121$, it will create an autonomous

[3] 20 times for each event

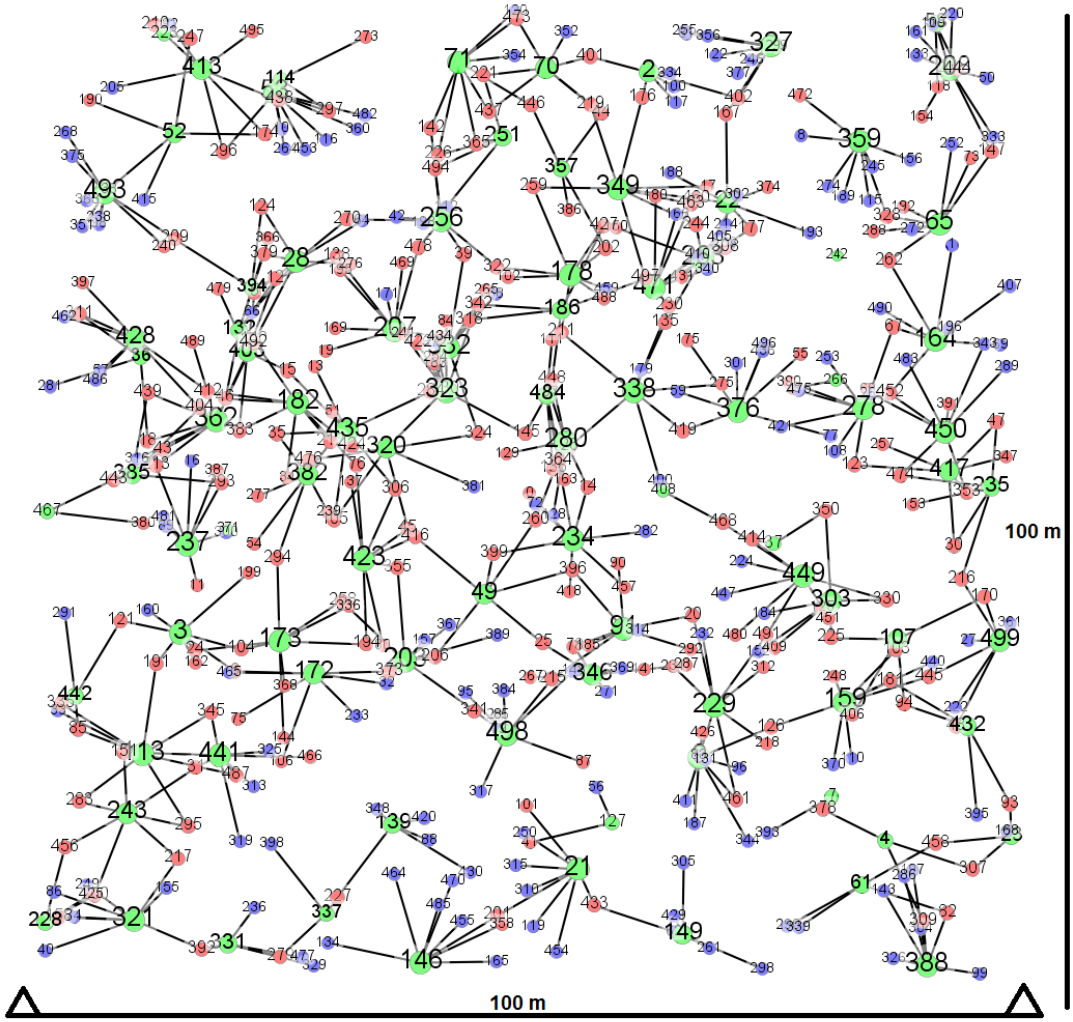


Figure 5.2: *A funeral in Iran — Simulated using WiDiSi*

group without any client, and will wait for a given amount of time to get connected to another part of the network via a bridge [4]. In the end, the node $n.121$ is able to connect to the network by means of node $n.22$, which is a client of group owner $n.153$. Since node $n.121$ was a group owner, this test also shows the correct behavior of MAGNET when two groups get close to each other (Event.4).

Event.2 The second event is the situation where a group owner leaves the proximity of its group, causing group termination. It is expected that the group owner and the clients were able to reconnect with the rest of the network. Figure 5.6 illustrates such a situation. In the first image, node $n.173$ is a group owner with eight P2P and legacy clients. The second image shows that the node $n.173$ is out of the radio range of its group and its clients now try to connect to another nearby groups ($n.97$ and $n.83$). The

[4] recall that if a zero-client group owner cannot get connected to the other part of the network using a bridge after a given amount of time (60 seconds in our example), It will try to remove the group and search for possible group connection or resending its request to the previous groups.

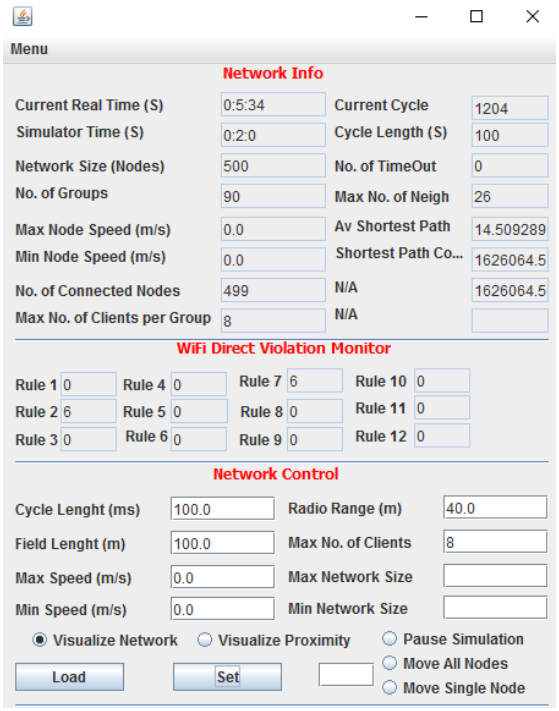
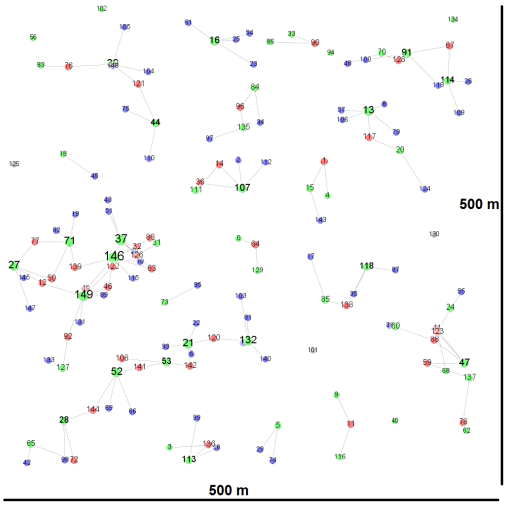


Figure 5.3: *A funeral in Iran — The network status*

(a) A group of customers in a shopping mall form a proximity-based Wi-Fi Direct network to find friends or search for a particular store in the proximity



(b) MAGNET behaviour in high mobility — Shopping Mall simulation on WiDiSi

Figure 5.4: *Case Study 2: Shopping mall*

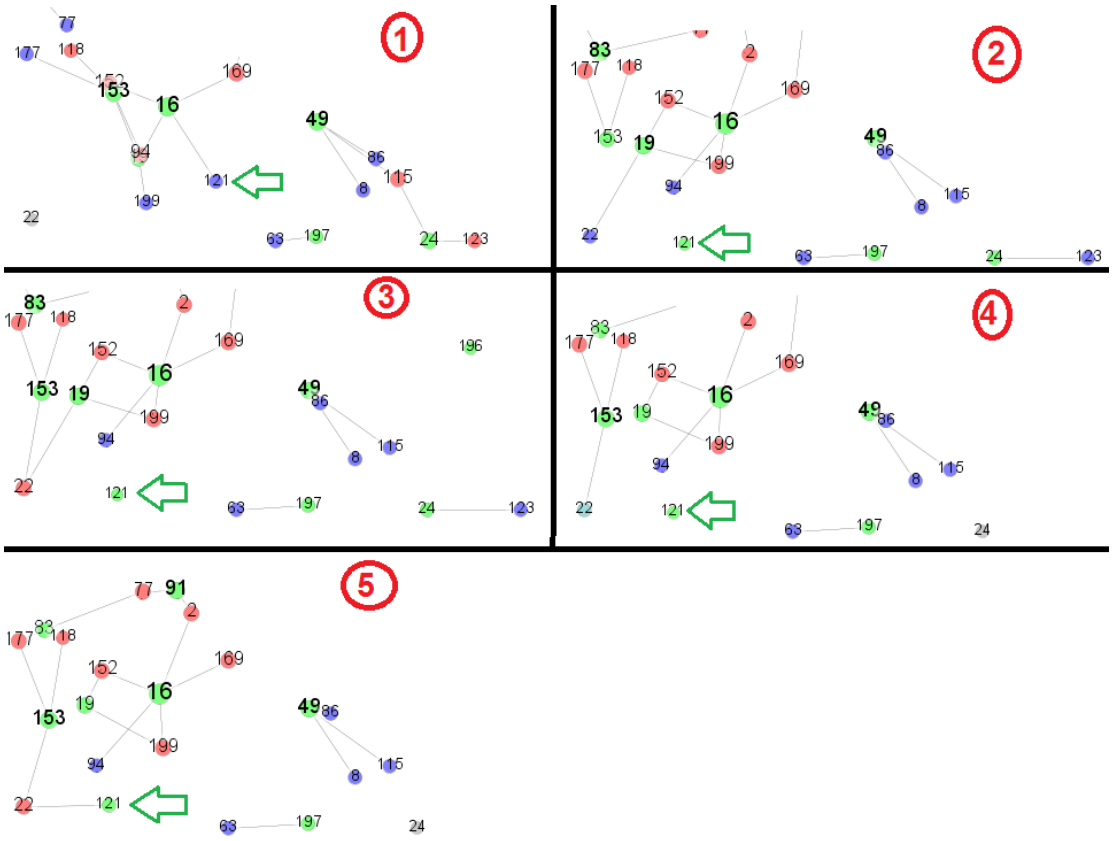


Figure 5.5: *MAGNET reaction to Event.1 and Event.4*

third image shows that the node $n.173$ now itself is inside the proximity of another group and becomes a P2P client of group owner $n.16$. By continuing on its way, $n.173$ is now in the proximity of another group and become a P2P client of another group ($n.49$). All the first clients of node $n.173$ are now connected as P2P, legacy or bridge to other groups. This example verifies the correctness of MAGNET behavior when facing group owner failure as well. We repeated this procedure for 20 random group owners, and MAGNET was able to reconnect the terminated groups.

Event.3 The third event is related to a situation where a bridge between two proximal groups fails or moves. It is expected that the two unconnected groups get connected again using another client. To test MAGNET in such situations, we move node $n.28$ in Fig. 5.7, which acts as a bridge between group owner $n.237$ and $n.111$. As it is shown in Fig. 5.7, the $n.28$ movement causes disconnection between two groups, however, group owner $n.237$ discovers this disconnection and tries to connect again to the same group via another client who is in the proximity of that group. In the end, the two groups will connect again via node $n.157$.

Event.5 In the last qualitative test, we simulate an environment where one or more devices face a saturated group. We assume that each group can host up to a maximum of **four**[5] clients in this test case. As Figure 5.8 illustrates, nodes $n.79$ and $n.128$ wanted

[5] We observed that Android version 5 and above limit the number of P2P clients in a Wi-fi Direct group to four.

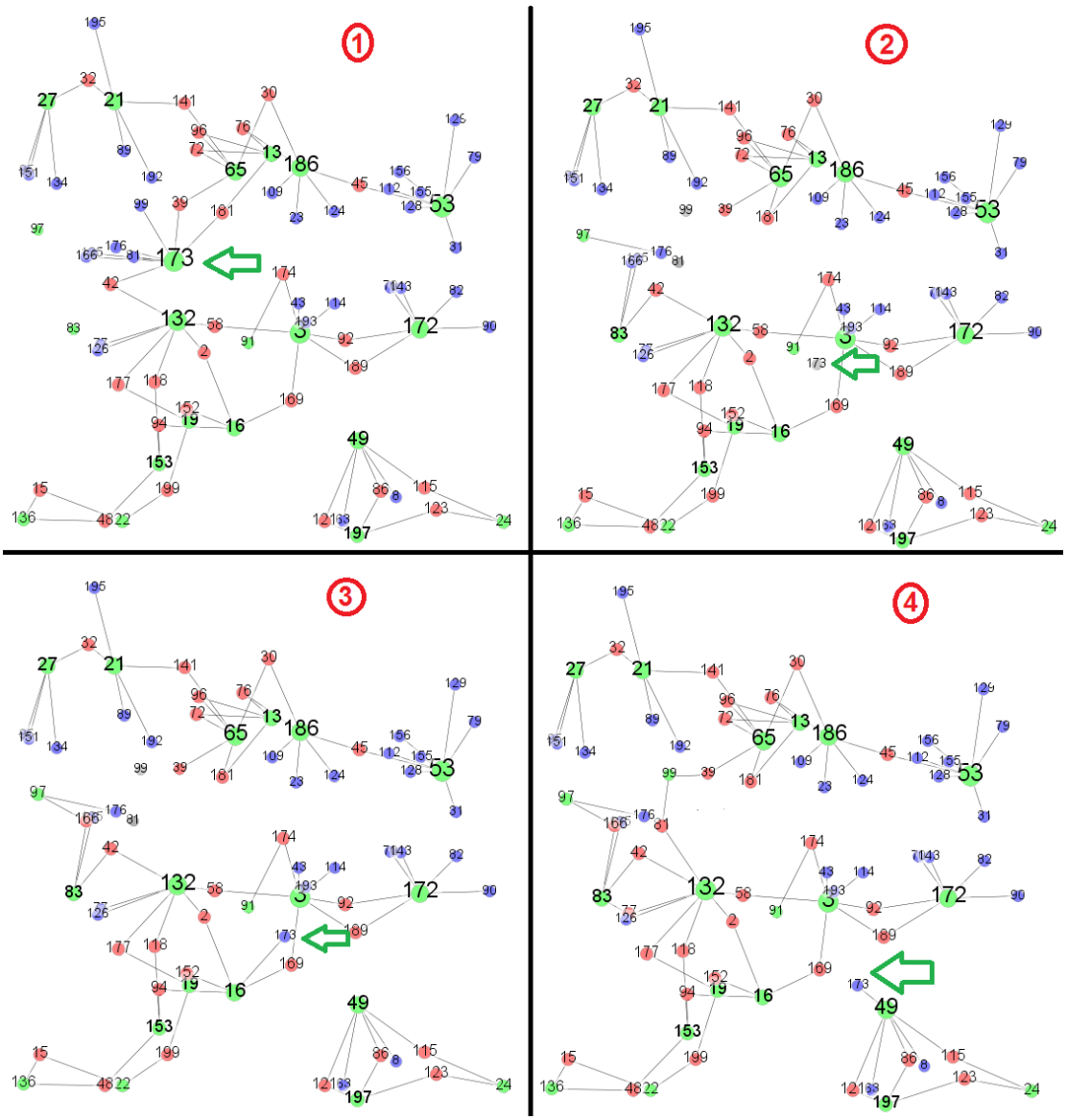


Figure 5.6: *MAGNET reaction to Event.2*

to join group owner $n.53$, however, the group owner had already hosted four other clients ($n.129$, $n.31$, $n.55$, $n.156$). Therefore, the group will reject any other requests, and, as a result, $n.79$ and $n.128$ do not have any chance to join $n.53$ as new P2P clients. It is expected that MAGNET connects these two nodes to the other part of the network. As illustrated in the third image of Fig. 5.8, $n.128$ and $n.79$ create a separate group and then they get connected to the network via $n.14$, which plays the role of a bridge in connecting groups $n.79$ and $n.0$.

5.2 Quantitative Evaluation

This section describes the quantitative evaluation of MAGNET. In section 5.1, we observed the correctness of MAGNET by testing it in different scenarios. In the following,

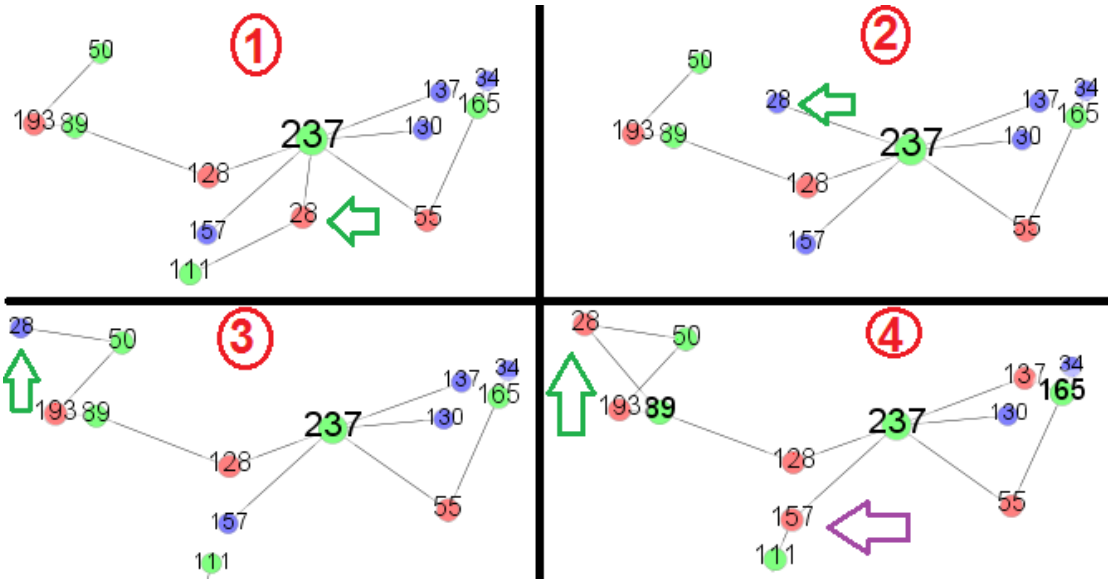


Figure 5.7: *MAGNET reaction to Event.3*

we evaluate the performance of MAGNET. We perform these tests using eight physical devices at SCUBE laboratory, eighteen physical devices at Samsung Laboratory, and in the simulation environment.

5.2.1 Case Study Three: Eight Physical Devices (SCUBE Lab)

This first experiment used eight devices running Android[6] (see Fig.5.11). MAGNET and the chat application were installed on every device, and we instrumented the app to log relevant data while in operation. We also synchronized all devices by means of the Internet time (with a one-second precision) and kept all the devices in the range of each other. Even if this last constraint was not mandatory, it allowed us to keep a simple configuration and to be sure that we were always able to create a solid network among the eight devices.

We conducted the experiment by using the heuristic routing algorithm. The number of devices was low, and the number of hops limited, thus a different routing algorithm would not have created significantly different results. Since our goal was to keep the devices connected in any possible scenario, we added random dynamism by randomly turning Wi-Fi connectivity on and off on the different devices. We wanted to measure the amount of time needed by MAGNET to re-organize the network and return to full operation. In particular we wanted to measure the following:

(a) The average time needed by the eight unconnected devices to discover the others and form a multi-group network;

(b) The average time needed by the network to recover when a group owner with one or more (legacy) clients leaves the network or dies;

[6]We used four HTC Nexus 9 tablets, three Samsung Galaxy S4s and one Samsung Galaxy S6 smartphones. These devices were running various versions of Android from 4.3 to 6.0.

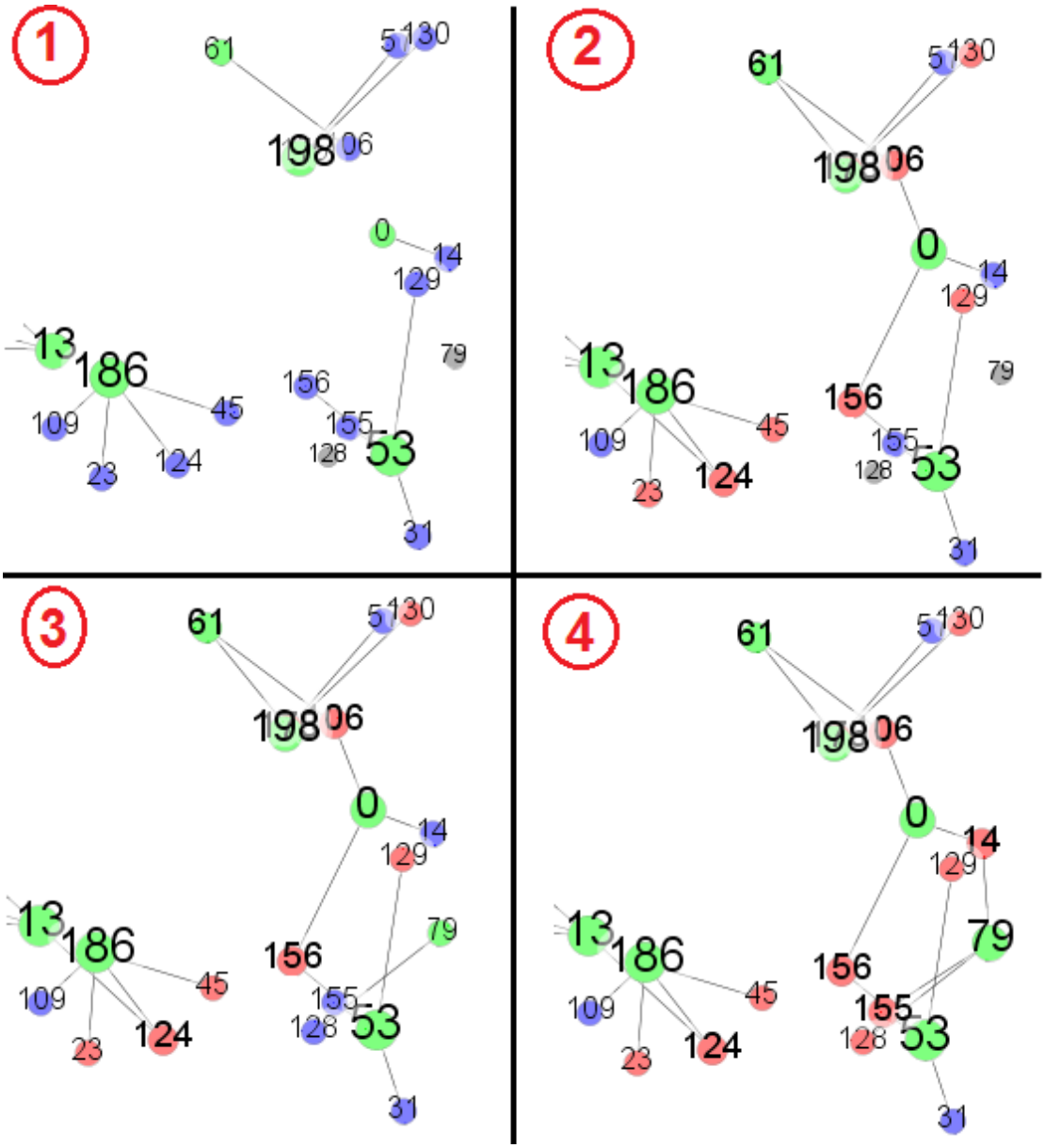


Figure 5.8: *MAGNET reaction to Event.5*

(c) The average time it takes a new device to enter the proximity of a group and connect to it;

(d) The average time needed by two groups to connect via a bridge;

(e) The average time needed by two groups to discover that a bridge is not available anymore and reconnect via another bridge;

(f) The average time needed by a new device to connect to the network when facing a saturated group;

We repeated the tests twenty times and computed the average delays illustrated in Fig.5.9. The highest delay refers to the case in which a group owner fails. This is why

owners must always be selected carefully, instead of simply using a random approach, to avoid long interruptions in the network's operation.

It is worth to mention that the delay values in Fig.5.9 are mainly related to fixed amount of time, which is set by the user (application developer). As noted earlier in Section 4.1, it is possible to change this amount of time in different application domains and, as a result, decrease the delay values in Fig.5.9; however, it may reduce the number of connected devices or groups as well. In other words, It is a trade off between network formation delay and network connectivity performance, and it is up to the application developer to choose the best amount of time for his application domain.

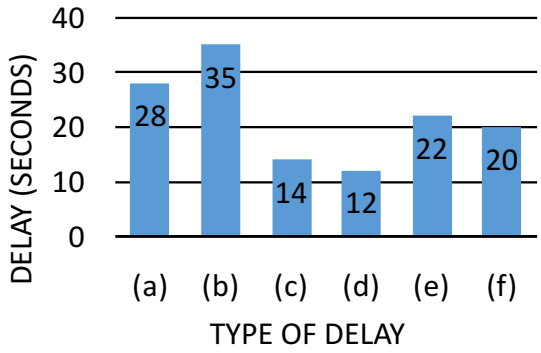


Figure 5.9: *Average delay for rehabilitating the network to working condition after various events (8 devices)*

Fig. 5.10 illustrates the relationship between the message delivery delay and the number of hops. Since the average delay for *zero-hop* deliveries is below one second, and we are not able to measure times smaller than a second, we used a default 500 ms for them.

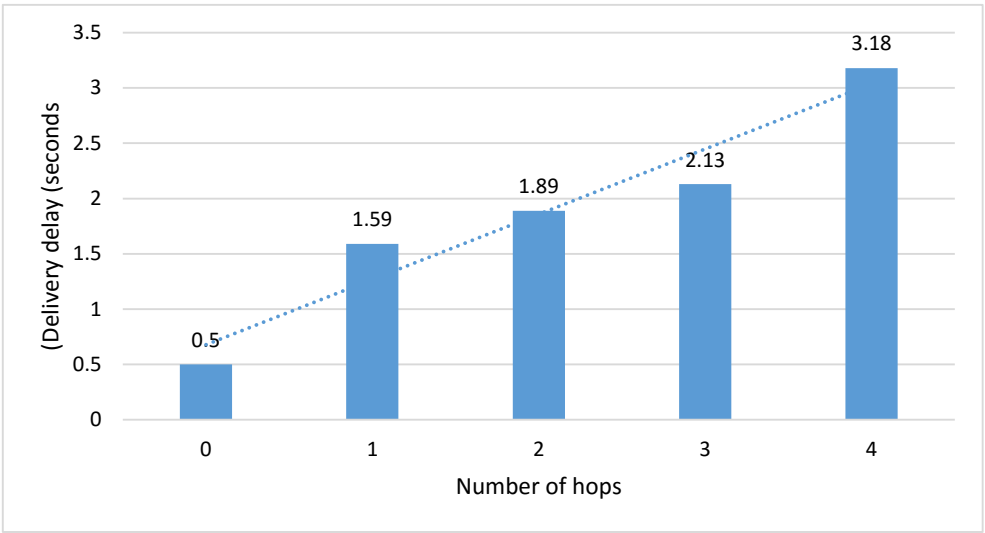


Figure 5.10: *The relationship between delays in message delivery and the number of hops (8 devices)*

To measure the message drop rate, we considered a network with three groups and two bridges. We considered this configuration as baseline to have paths with a dif-

ferent number of hops (messages have to pass from zero to four hops to reach their destination) and different communication types (messages exchanged between group owners, clients of the same group, clients of different groups, etc.). Thus, we first kept the network unmodified to calculate the message drop rate in a stable situation with a maximum number of four hops per message (Fig. 5.11). After some experiments, we decided that each device had to send a message to all the other devices in the network every 5 seconds. Out of a total number of 672 sent messages, 642 messages were delivered successfully, for a drop rate of only 4.46%[7]. Since this value is measured in a static network, it can be regarded as the minimum drop rate. To measure the drop rate in a dynamic context, we turned the Wi-Fi receivers on and off on the different devices. Out of the 9560 messages we sent, 5674 messages were delivered successfully, with a drop rate of 40.9%. This high drop rate is due to the fact that the source device does not know whether the destination is connected to the network at the time of sending the message. This situation requires the adoption of a timeout mechanism to re-send the message if the response does not arrive within a given time interval.

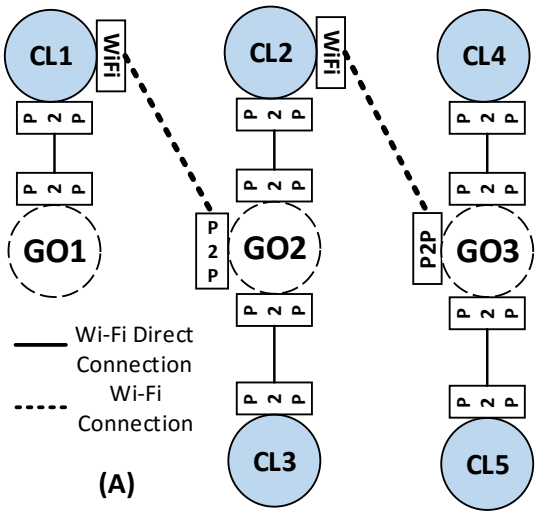


Figure 5.11: *The topology of the network for measuring the baseline for message drop rate*

5.2.2 Case Study Four: Eighteen Physical Devices (Samsung Lab)

Since gathering a higher number of physical devices at a university laboratory was difficult, we performed the second experiment with eighteen tablets in a Samsung laboratory in Milan. These included six Galaxy Note 10s with Android version 5.1.1 and twelve Galaxy Tab 4s with Android version 5.0.2 (Fig. 5.12).

Since we did not have access to a USB hub with the required amount of ports, we used Google Drive to send installation files and collect debugging logs. Similar to the previous case, we synced the clocks on each device with the Internet time and kept all devices in range to be sure that we could create a solid network. The maximum number of clients per each Wi-Fi Direct group was limited to four by Android[8].

[7] This drop rate comes from the uncertainty in UDP transmissions and is not related to MAGNET's operation. It can thus be considered as a baseline drop rate.

[8] This value was fixed by the Android OS

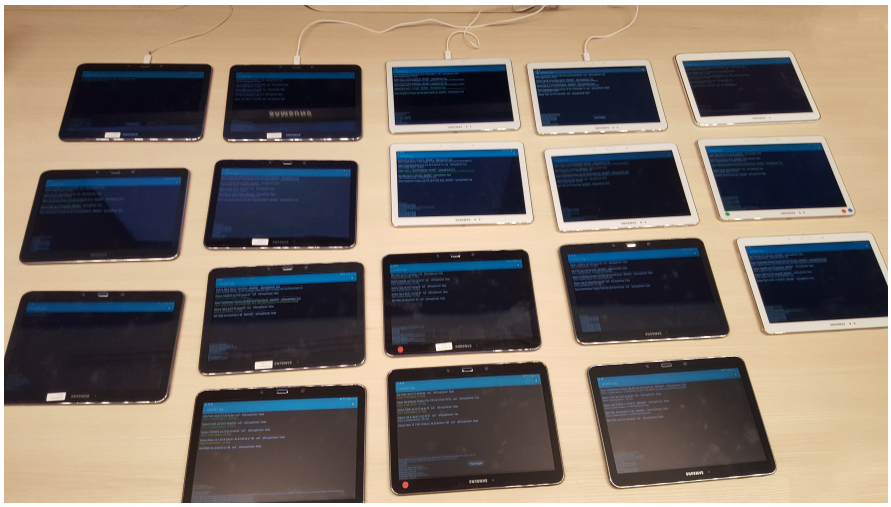


Figure 5.12: *Experiment on eighteen devices at Samsung Lab*

Similar to the previous test on eight devices, we conducted the experiment by using the heuristic routing algorithm. To add dynamism to the network, we used two techniques: 1) turning Wi-Fi connectivity on and off on the different devices, and 2) guiding the application to remove its current groups and start searching again for a new connection. The former mechanism mimics the situation where a device dies because of the battery. In this situation, a device needs to start the Wi-Fi again, which adds some delays to the group/network formation. The last mechanism mimics the situation where a device moves from the proximity of one group and enters the proximity of another group. In this situation, there is no need to wait until the Wi-Fi radio is activated. The combination of these two mechanisms helps to generate more realistic scenarios. For the first experiment, we wanted to measure the average delay mentioned in section 5.2.1. We repeated the test ten[9] times and computed the average delay, illustrated in Fig. 5.13.

By comparing the values in Fig. 5.9 that are related to the experiment on eight devices with the values in Fig. 5.13 that are linked to the experiment on eighteen devices, we can state the following conclusions:

1. The highest delay still belongs to a situation where a group owner dies.

2. Group/network bootstrapping delays in the experiment with eighteen devices are clearly greater than the experiment with eight devices. This increase was clearly observed during the experimentation. The obvious reason behind that is the group saturation. As mentioned earlier in chapter 4, the best group formation is when the devices with the highest intention values become owners, and all groups reach the maximum number of clients. Since the number of devices is high, we end up with the greater number of groups that reach their limits. Therefore, when a new device sends its request to join a group, the chance for the request to get accepted is lower, since most groups have already reached their maximum client capacity. In the occurrence of this event, the new devices should repeat the group bootstrapping

[9]Since access time to the Samsung laboratory in Milan was limited, we were not able to have more trials. However, even this number of trials is enough to show the performance of MAGNET

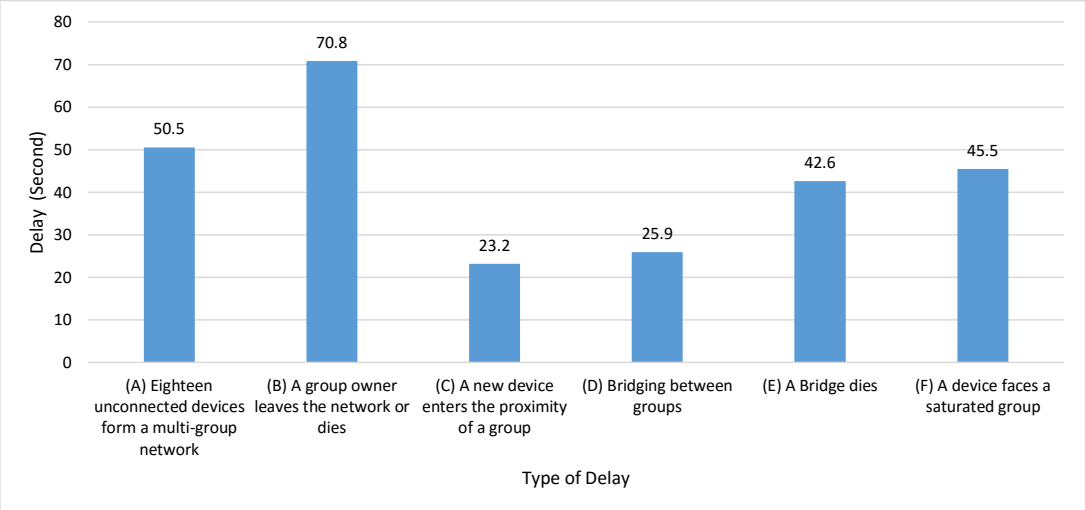


Figure 5.13: *Average delay for rehabilitating the network to working condition after various events (18 devices)*

procedure, which increases the delay. This explanation can be extended to all delay items in Fig. 5.13.

3. The delays related to the experiment on eighteen devices are almost twice the ones on eight devices. One could conclude that these delays would increase linearly to correspond with the number of devices. However, this conclusion may not always hold true. Many factors affect the group/network bootstrapping delays (such as signal congestion, group capacity, network topology, environmental noise, and the distance between devices). Therefore, without further experiments with a higher number of devices, such statements are not correct.

Fig. 5.14 illustrates the relationship between the message delivery delay and the number of hops. The stated delays are an average of ten thousand delivered messages. The experiment with eighteen devices clearly outperforms the experiment on eight devices in terms of delivery delay per hop. For instance, the delivery delay for four hops in this experiment is 2.39 seconds. That is much better than 3.18 seconds in the experiment on eight devices. However, we can argue that this decrease in the delay is not related to MAGNET, since these values are measured in a static network. Other possible reasons for such a difference could be the better hardware of the tablets, or the consistency of the platforms (all devices were Samsung tablets with Android 5).

Fig. 5.15 illustrates the maximum and the minimum values for message delivery delay per hop. As can be seen from this figure, the minimum values for one to seven hops are below one second. Moreover, the highest value is related to three hops. However, these numbers are not meaningful unless we also provide the standard deviation for each set of data[10] (Fig. 5.16). Standard deviation shows us how much closer to the mean value the actual data points are. Therefore, the most consistent data set in our experiment belongs to zero hops and eight hops, where most of the data points are very close to the mean, and the least consistent data set is associated with the two hops,

[10]Each set contains one thousand data points

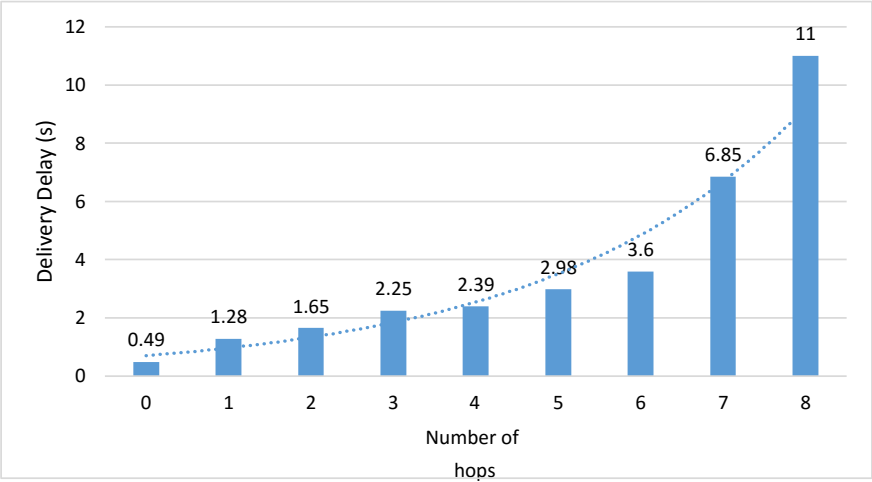


Figure 5.14: *The relationship between delays in message delivery and the number of hops (18 devices)*

where most of the data points are far from the average. One reason behind this is that the chance to reach devices over eight hops is lower than two hops. Those packets that can reach the destination are probably have traveled the same route with the same delay.

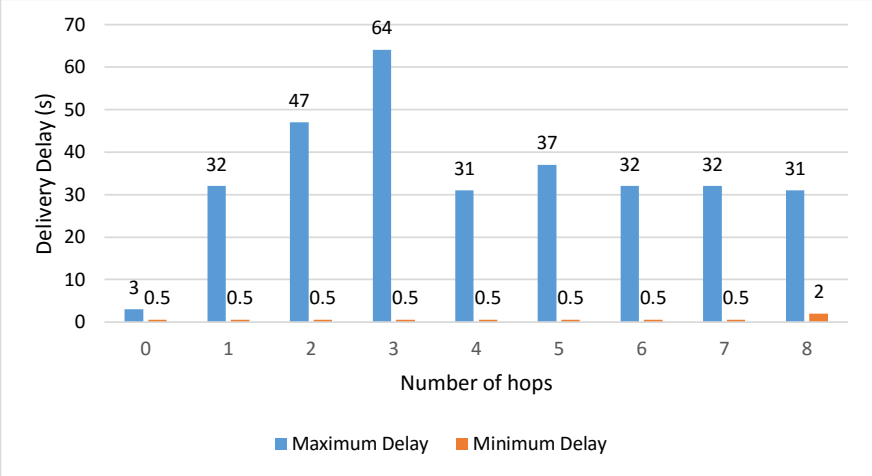


Figure 5.15: *The maximum and the minimum message delivery delays per hop (18 devices)*

To measure the drop rate in a dynamic context, we turned the Wi-Fi receivers on and off on the different devices. In addition to this, we guided the application to exit a group and restart the group/network bootstrapping procedure. We guided the application on each device to broadcast a message to all other devices inside the network every 10 seconds[11]. Out of the $11,304$ messages we sent, $7,788$ messages were delivered successfully, with a drop rate of 31.1%. In this experiment, two factors are working against each other. On the one hand, the higher number of hops will increase the drop rate, since each hop utilizes an unreliable UDP transmission. On the other hand, the greater number of devices will result in a higher number of paths between sender and receiver, which leads to a better chance for a message to reach the destination. As

[11]This number is chosen to prevent message congestion

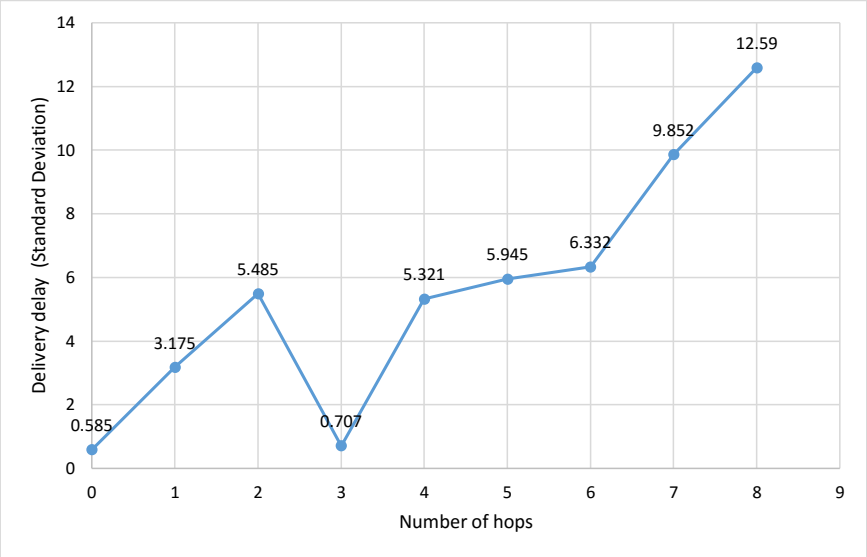


Figure 5.16: *The standard deviation of delay values per hop (18 devices)*

can be seen, the latter factor is more important than the former one. In an experiment with fewer hops and a smaller number of paths, the drop rate was 40.9%, and in this experiment with more hops and a larger number of paths, the drop rate is reduced to 31.1%.

5.2.3 Case Study Five: Simulations on WiDiSi

Although the use of physical devices provides more accurate results and can help better explain the behavior of our middleware in real scenarios, it comes with diverse problems. Clearly, the first issue is the availability of the (high) number of required devices, which may be costly or even impossible in a university research laboratory. The other obvious problem is that the complexity of designing meaningful experiments increases with the number of used devices. Devices should move according to predefined patterns, enter or leave the system at will, or fail due to battery issues. Moreover, a powerful source of noise in the devices' proximity may jam the Wi-Fi Direct signals and cause disruption.

These are some of the reasons why we decided to use our Wi-Fi Direct simulator, called *WiDiSi* [27], to conduct some experiments with large, complex scenarios. *WiDiSi* allowed us to simulate large-scale Wi-Fi Direct networks with various mobility patterns.

The experiment on the simulator considered the same chat application introduced above, but this time we used the AODV routing algorithm. The metric of interest is now the degree of network connectivity, defined as (1) the average number of devices that are connected to a group (*group connectivity*), and (2) the average number of devices each device has access to (*network connectivity*).

Even if we claim that MAGNET can always keep all devices connected to each other, some devices may be temporarily out-of-range. For instance, in Fig. 5.17, the different clusters[12] of the network are out of range of each other and there is not possibility to

[12] A cluster is a set of connected devices that are not connected to the other part of the network (probably because they are out

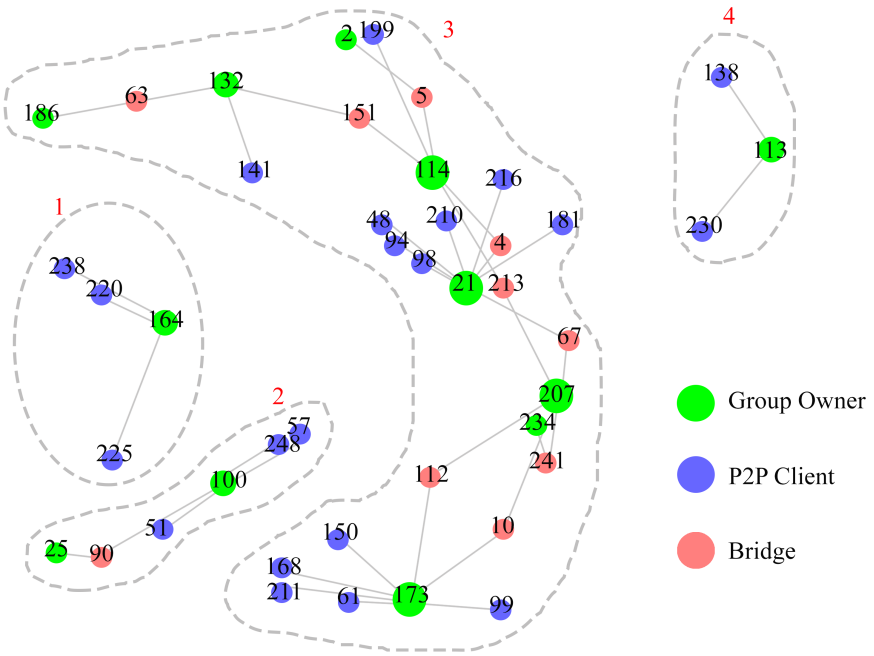


Figure 5.17: *A section of the network with four separate clusters*

connect them to each other via one or more bridges.

Our experiments considered a dense test scenario with 250 devices randomly distributed in a $1km^2$ open space (A section of the network is illustrated in Fig. 5.17). The theoretical radio range of the devices is 200 meters; however, because of the usual congestion, we set it to 80 meters. Although it is still possible that a device go of-out-range and thus remains unconnected, the chance is low in this scenario. Fig. 5.18 illustrates the relationship between the speed with which devices move and the two metrics of interest.

Fig. 5.18 shows that when they move at a given speed and when the network is static, MAGNET can connect almost all the devices to a group (*group connectivity* would be 100%) and that, on average, each device has access to some 71% of the other devices in the network (*network connectivity*). When the devices start moving, these figures decrease. For example, when devices move at a normal human speed (i.e., 3 m/s), *group connectivity* becomes 94% and *network connectivity* drops to 42%. Since devices move from one group to another, the use of a store-and-forward technique can provide message delivery even if different subnets are not physically connected to each other in that moment. For example, if node 100 of Fig. 5.17 stored the messages of its current group and moved to the proximity of node 173, it could forward stored messages to the devices of that cluster.

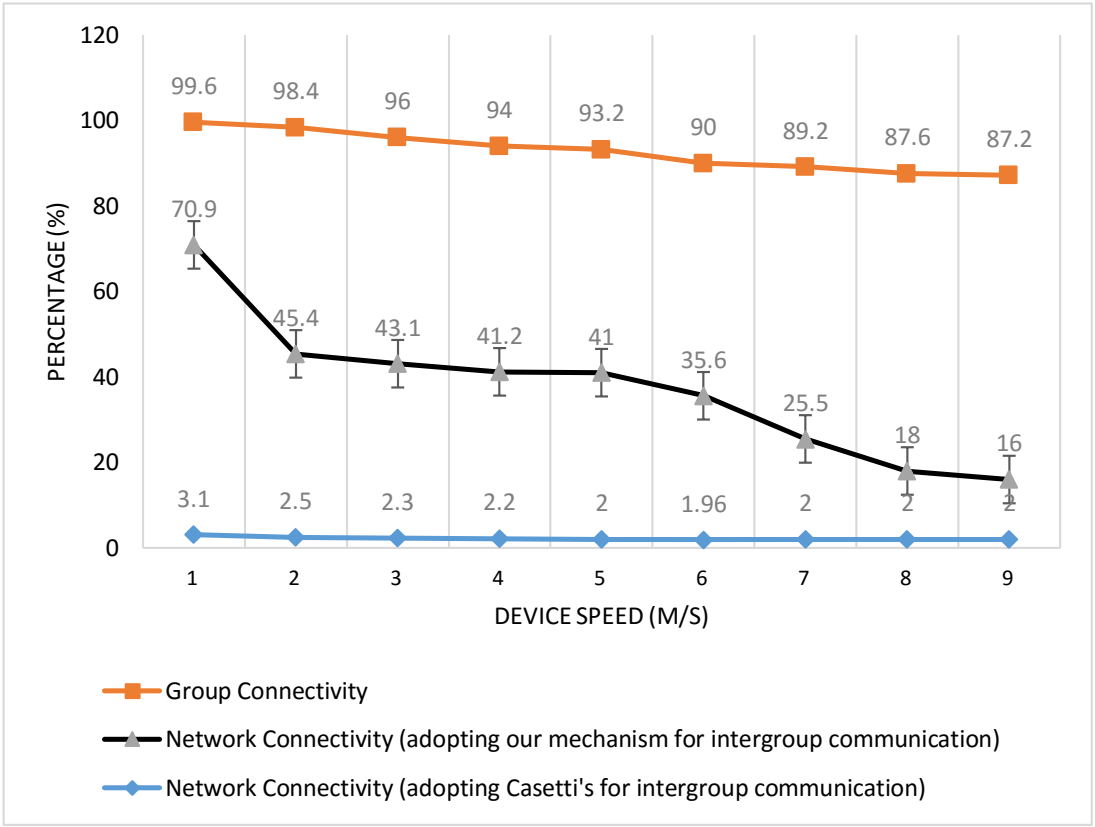


Figure 5.18: *Relationship between devices' mobility and network connectivity*

of range of the other parts).

5.3 Threats to validity

Internal validity The experiments on the physical devices (Fig. 5.9) illustrate the relationship between the type of movement/failure and the time required for the network to reform. If some clients die at the same time the delay for network reformation is not the same as when only a bridge dies or moves. This is why we kept all the devices in the network static, except for the one being tested, to be sure that the result was only affected by that single device and not by any other event in the network. Moreover, all devices in a group keep track of the configuration of the group and in the future they can bypass the authentication process. As a result, while configurations are saved, the time needed for forming groups decreases. To mitigate this problem, we did not consider sequences of events of the same type one after the other, but we randomly generated sequences of events of different types.

External validity First of all, we need to comment on the *generality* of results obtained with "only" eight devices. Since decisions in MAGNET are taken locally without any global knowledge of the network, we are confident that obtained results can be generalized. Note that our experiments were carried out in a real context (our laboratory) with many different sources of noise at the same frequency (Wi-Fi networks and lots of Bluetooth and ZigBee devices). As long as the density of devices in the space does not preclude regular Wi-Fi communication, the number of devices does not effect the behavior of MAGNET.

It is true however that more experiments with many more devices are needed. We could only make use of an in-house built simulator, and thus we could only resemble a "reasonable" behavior. More accurate models of the MAC and PHY layers would have allowed us to further refine obtained results. We need to study in more detail how MAGNET behaves when the drop package rate at the MAC layer becomes too high.

CHAPTER 6

Conclusion and Future Directions

We started the thesis by describing the Internet of Things and the various types of communications in this area. We presented different research questions that related to the limitations in exploiting Wi-Fi Direct in large-scale dynamic application scenarios and the lack of a simulation environment for Wi-Fi Direct. In this chapter, we conclude the thesis by indicating the solutions provided for each research question. We also discuss related work and provide some final considerations.

6.1 Answers to Research Questions

This section restates the research questions and examines the contribution of this thesis for each one.

6.1.1 Research Question One [Q.1]

The question: *Is Wi-Fi Direct a suitable candidate to enable proximity-aware interaction of smart things in an IoT scenario?*

The answer: To answer the first question [Q.1], we performed a survey of the most promising middleware technologies and wireless communication protocols that can support (future) proximity-based applications. In the following, we give a summary of this review.

1. We discussed Bluetooth, Wi-Fi Direct, 6LowPan, Thread, and LTE Direct in this survey. In addition to wireless communication protocols, a developer may benefit from middleware infrastructures to perform more sophisticated tasks. The use of middleware would also provide better interoperability, reusability, and security compared to using embedded APIs. In this survey, we discussed AllJoyn, IoTivity,

and NearBy as the most promising middleware technologies in this application domain. The communication in these middleware infrastructures is independent of the transport layer, and this means that it can be changed transparently from the application's perspective.

2. We claimed that this overview could provide a first step in helping developers choose the most appropriate technology stack for their application. This choice will largely depend on the functional and non-functional requirements of the applications, the available devices, and the type of proximity-based discovery and communication required. Complex applications can also benefit from choosing a combination of these technologies. For example, LTE-Direct can be used for scalable discovery and Wi-Fi Direct for high-speed communication.

3. We argued that Wi-Fi Direct is a suitable candidate for our working scenario regarding speed, range, power consumption, stability, and wide availability in smartphones compared to the other communication protocols. The low radio range of Bluetooth, the unavailability of 6LowPan and Thread on smartphones, and the centralized control of LTE-Direct and its intrinsic inability for communication in all situations are the most significant drawbacks of these protocols for their use in enabling the proximity-aware interaction of smart things in an IoT scenario.

4. We have also emphasized the Wi-Fi Direct limitations in IoT scenarios. We stated that the number of devices and services that a Wi-Fi Direct device can discover and connect to are very limited. Moreover, Wi-Fi Direct cannot handle mobility of devices. We concluded that a middleware infrastructure is needed to address these limitations in a large-scale mobile environment.

5. Finally, it is worth mentioning that the success of (future) proximity-based applications will not only depend on the adopted technology, but also on the added value they will be able to offer to their users.

6.1.2 Research Question Two [Q.2]

The question: *How is it possible to remove Wi-Fi Direct intrinsic limitations and exploit it to connect a large number of devices?*

The answer: To respond to the second question [Q.2], we proposed MAGNET, a Wi-Fi Direct-based middleware for the interaction of Android devices in proximity. The thesis discusses both the overall concepts behind the MAGNET and the key design and implementation decisions to make things work and find a proper solution to the well-known reliability and scalability issues that hamper the broad adoption of Wi-Fi Direct. We proposed the concept of multi-group management and communication. Instead of hosting all devices in one group, which is practically impossible, we proposed creating several independent Wi-Fi Direct groups and then connecting groups to each other via some devices that play the role of a bridge between groups. In this way, any number of devices can become connected to each other. The MAGNET manages the group's formation and decides the devices' roles (group owner, client or bridge) by considering relevant criteria. We also proposed a novel solution for intergroup routing and communication to enable multi-hop communication inside a Wi-Fi Direct network.

6.1.3 Research Question Three [Q.3]

The question: *How is it possible to manage mobility in a Wi-Fi Direct network to offer a stable connection to the application?*

The answer: Mobility management is another feature of the proposed middleware infrastructure. MAGNET autonomously tries to keep the multi-hop connectivity between devices and also provides proper interfaces to let the user configure some key parameters, given the context of the operation. In the case of devices entering/leaving/-failing, the MAGNET restart group bootstrapping or intergroup connection procedure considers user preferences, to reestablish a broken connection autonomously.

6.1.4 Research Question Four [Q.4]

The question: *How is it possible to test and evaluate the effectiveness of proposed solutions in realistic large-scale dynamic situations?*

The answer: The evaluation of the proposed solution has been tested on real Android devices. However, due to practical limitations of testing on a large number of physical mobile devices, we introduced a novel Wi-Fi Direct simulation environment, called WiDiSi. WiDiSi is a prototype simulator for Wi-Fi Direct. The simulator provides the same APIs as the Android implementation of the protocol, which results in extreme portability between WiDiSi and real devices. The evaluations conducted so far confirm that WiDiSi is capable of creating and visualizing a large-scale Wi-Fi Direct network.

The evaluation we carried out on MAGNET, both through our simulator and a (limited) set of devices, suggests that the middleware can provide reliable communication between large numbers of Android devices in dynamic contexts.

6.2 Future Directions

This thesis has opened many interesting directions for the future. The focus of this work was to remove known Wi-Fi Direct limitations. However, it is possible to exploit other well-known communication protocols such as Bluetooth. Moreover, we need to investigate strategies for multi-protocol communication, and, probably, to enable dynamic and intelligent protocol selection for discovery and communication. Our plan is to keep refining the implementation of our middleware, ameliorate it with additional features such as a store-and-forward mechanism, and addressing security/privacy issues, and conduct a more thorough assessment of a wider set of physical devices.

Furthermore, we are eager to ameliorate the Wi-Fi Direct simulator with additional features such as persistent group formation, power management modeling, UPnP service advertisement/discovery, and encryption message modeling. In addition to this, the simple exploited models for MAC, PHY, and the free space channel are not accurate enough for some application domains. More detailed and complex models are needed to produce a more accurate simulation. Moreover, it is necessary to carry out a more complete and comprehensive evaluation by comparing the results produced by WiDiSi against those obtained in a real network of Android devices.

Bibliography

[1] A walk through Internet of Things (IoT) basics. https://opentechdiary.wordpress.com/category/iot-basics/.

[2] Discover Wi-Fi Direct, Portable Wi-Fi that goes with you anywhere. http://www.wi-fi.org/discover-wi-fi/wi-fi-direct.

[3] Scatternet - part 1, baseband vs. host stack implementation - white paper. Technical report, Ericsson, Inc., June 2004.

[4] Wi-Fi Peer-to-Peer (P2P) Technical Specification (Version 1.1). *Wi-Fi Alliance Technical Committee P2P Task Group*, pages 1–159, June 2010.

[5] more than 50 billion connected devices — taking connected devices to mass market and profitability. Technical Report 284 23-3149 Uen, Ericsson, February 2011.

[6] Wi-fi performance benchmark testing: Aruba networks ap-135 and cisco ap3602i. Technical report, Aruba Proof of Concept (PoC) Lab, Aruba Networks, Inc., Sunnyvale, California, June 2012.

[7] Black book: 802.11ac wi-fi benchmarking. Technical report, ixia, Inc., June 2014.

[8] Compass.to demo at CES 2015 shows future of LTE Direct, 2015. https://medium.com/@CompassTo/.

[9] Discrete-Event Network Simulator for Internet Systems. NS-3 Consortium, 2015. https://www.nsnam.org/.

[10] Gephi toolkit javadoc. Gephi, 2015. https://gephi.org/docs/toolkit/.

[11] Nearby: Build simple interactions between nearby devices and people. . Google Inc., 2015. https://developers.google.com/nearby/.

[12] The Open Simulation Platform for Intelligent Transport System (ITS) Services. iTETRIS, 2015. http://www.ict-itetris.eu/.

[13] Wifip2pmanager. Google Inc., 2015. http://developer.android.com/reference/android/net/wifi/p2p/WifiP2pManager.html.

[14] CQL: Contextual Query Language Specification. SRU standard, 2016. http://www.loc.gov/standards/sru/cql/spec.html.

[15] Geography - Quick Guide. tutorialspoint, 2016. http://www.tutorialspoint.com/geography/geography_quick_guide.htm.

[16] Lead the Next Generation of Proximity Services. Qualcomm Inc., 2016. https://ltedirect.qualcomm.com/.

[17] What is Thread? Thread Group, Inc., 2016. http://www.threadgroup.org/.

[18] 3GPP. 3rd generation partnership project; technical specification group radio access network; study on lte device to device proximity services; radio aspects(release 12). Technical report, March 2014.

[19] A. Levy. 200ms: The Magical Number for Faster Response Times. Crittercism, Inc, 2014. http://www.crittercism.com/blog/200ms-the-magical-number-for-faster-response-times.

[20] AllSeen Alliance. Alljoyn framework. https://allseenalliance.org/framework. (Accessed on 09/23/2016).

[21] Wi-Fi Allieance. Wi-Fi Peer-to-Peer (P2P) Technical Specification (Version 1.1). Technical report, June 2010.

[22] M.S. Amin, F.A. Bhuyan, M.L. Rahman, and M.S. Amin. Bluetooth Scatternet Formation Protocol: A Comparative Performance Analysis. In *Asia-Pacific Conference on Communications*, pages 1–5, Aug 2006.

[23] S. Andreev, O. Galinina, A. Pyattaev, K. Johnsson, and Y. Koucheryavy. Analyzing Assisted Offloading of Cellular User Sessions onto D2D Links in Unlicensed Bands. In *IEEE Journal on Selected Areas in Communications*, volume 33, pages 67–80, Jan 2015.

[24] SuperBeam App. Wifi direct share. https://play.google.com/store/apps/details?id=com.majedev.superbeam. (Accessed on 09/23/2016).

[25] Scott Armitage. Overview of ieee 802.11u. Technical report, Loughborough University, Loughborough, England, March 2012.

[26] Somil Asthana. Bluesim : Bluetooth simulation tool for ns. Technical report, November 2005.

[27] L. Baresi, N. Derakhshan, and S. Guinea. WiDiSi: A Wi-Fi Direct Simulator. In *Proceedings of the IEEE International Conference on Wireless Communications and Networking*, pages 1–7, 2016.

[28] S. Basagni, R. Bruno, G. Mambrini, and C. Petrioli. Comparative Performance Evaluation of Scatternet Formation Protocols for Networks of Bluetooth Devices. In *Springer-Verlag Wireless Networks*, volume 10, pages 197–213, March 2004.

[29] P. Bellavista, A. Corradi, and C. Giannelli. Mobility-aware Middleware for Self-Organizing Heterogeneous Networks with Multihop Multipath Connectivity. *IEEE Wireless Communications*, 15(6):22–30, December 2008.

[30] P. Bellavista, A. Corradi, and C. Giannelli. Differentiated Management Strategies for Multi-Hop Multi-Path Heterogeneous Connectivity in Mobile Environments. *IEEE Transactions on Network and Service Management*, 8(3):190–204, September 2011.

[31] Bluetooth SIG. *BLUETOOTH SPECIFICATION Version 4.1*, 12 2013.

[32] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano. Device-to-device Communications with Wi-Fi Direct: Overview and Experimentation. *IEEE Wireless Communications*, 20(3):96–104, June 2013.

[33] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano. Device-to-device communications with wi-fi direct: overview and experimentation. In *IEEE Wireless Communications*, volume 20, pages 96–104, June 2013.

[34] C. Casetti, C. F. Chiasserini, L. C. Pelle, C. D. Valle, Y. Duan, and P. Giaccone. Content-Centric Routing in Wi-Fi Direct Multi-Group Networks. In *Proceedings of the* 16^{th} *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–9, 2015.

[35] P. Chaki, M. Yasuda, and N. Fujita. Seamless Group Reformation in WiFi Peer to Peer network using dormant backend links. In *Proceedings of the* 12^{th} *IEEE International Conference on Consumer Communications and Networking*, pages 773–778, 2015.

[36] Ashok Babu Channa. IoTivitiy Architecture. Linux Foundation, 2016. https://wiki.iotivity.org/architecture.

[37] Tom Cocagne. Dbus overview, 2012. https://pythonhosted.org/txdbus/dbus_overview.html.

[38] Naser Derakhshan. WiDiSi: A Wi-Fi Direct Simulator. GitHub, 2015. https://github.com/nasser1941/WiDiSi.

[39] Google Developers. Nearby. https://developers.google.com/nearby/. (Accessed on 09/25/2016).

[40] E.W. Dijkstra and C.S. Scholten. Termination Detection for Diffusing Computations. In *Elsevier Information Processing Letters*, volume 11, pages 1–4, 1980.

[41] A. Djajadi and R.J. Putra. Inter-Cars Safety Communication System Based on Android Smartphone. In *Proceedings of the IEEE International Conference on Open Systems*, pages 12–17, 2014.

[42] Daniel J. Dubois, Yosuke Bando, Konosuke Watanabe, and Henry Holtzman. ShAir: Extensible Middleware for Mobile Peer-to-peer Resource Sharing. In *Proceedings of the* 9^{th} *ACM Joint Meeting on Foundations of Software Engineering*, pages 687–690, 2013.

[43] R. A. Ferreira, M. K. Ramanathan, A. Grama, and S. Jagannathan. Randomized Protocols for Duplicate Elimination in Peer-to-Peer Storage Systems. In *IEEE Transactions on Parallel and Distributed Systems*, volume 18, pages 686–696, 2007.

[44] Open Connectivity Foundation. Iotivity. `https://www.iotivity.org/`. (Accessed on 09/23/2016).

[45] C. Funai, C. Tapparello, and W.B. Heinzelman. Supporting Multi-Hop Device-to-Device Networks Through WiFi Direct Multi-Group Networking. In *ACM Transactions on Computing Research Repository*, volume abs/1601.00028, pages 1–7, 2016.

[46] D. Gomez, E. Rodriguez, R. Aguero, and L. Munoz. Reliable Communications Over Lossy Wireless Channels by Means of The Combination of UDP and Random Linear Coding. In *Proceedings of the* 19^{th} *IEEE International Symposium on Computers and Communications*, pages 1–6, 2014.

[47] Damien Guard. Bluetooth LE device simulator for iOS. GitHub, 2014. `https://github.com/AttackPattern/BlueSim`.

[48] J. Härri, P. Bellavista, L. Foschini, and R. Blokpoel. Extending the iTETRIS platform for Smartphone Sensing and Communication Simulation. In *Proc. of the 5th European Conference on Transport Research Arena*, April 2014.

[49] P.A. Hosein, J.T. Walton, and M. Athans. Dynamic Weapon-Target Assignment Problems with Vulnerable C2 nodes. Technical Report LIDS-P-1786, Massachusetts Institute of Tech Cambridge Lab for Information and Decision Systems, 1998.

[50] Apple Inc. Multipeerconnectivity. `https://developer.apple.com/reference/multipeerconnectivity`. (Accessed on 09/23/2016).

[51] Qualcomm Inc. White paper: LTE Direct Trial. Technical report, February 2015. `https://www.qualcomm.com/media/documents/files/lte-direct-trial-white-paper.pdf`.

[52] Apurva Kumar. BlueHoc: Bluetooth Performance Evaluation Tool. IBM Corporation, 2001. `http://bluehoc.sourceforge.net/`.

[53] Gyu Myoung Lee, Jungsoo Park, Ning Kong, and Noel Crespi. The Internet of Things - Concept and Problem Statement. `https://tools.ietf.org/html/draft-lee-iot-problem-statement-00`.

[54] Myungjae Lee. Soft Sensor Manager. Linux Foundation, 2015. `https://wiki.iotivity.org/soft_sensor_manager_for_linux`.

[55] H. Luo, X. Meng, R. Ramjee, P. Sinha, and L. Li. The Design and Evaluation of Unified Cellular and Ad-Hoc Networks. In *IEEE Transactions on Mobile Computing*, volume 6, pages 1060–1074, Sept 2007.

[56] N. Maalel, E. Natalizio, A. Bouabdallah, P. Roux, and M. Kellil. Reliability for Emergency Applications in Internet of Things. In *Proceedings of the* 9^{th} *IEEE International Conference on Distributed Computing in Sensor Systems*, pages 361–366, May 2013.

[57] K. Mikhaylov. Simulation of network-level performance for bluetooth low energy. In 25^{th} *IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC)*, pages 1259–1263, September 2014.

[58] G. Miklos, A. Racz, Z. Turanyi, A. Valko, and P. Johansson. Performance aspects of bluetooth scatternet formation. In *First Annual Workshop on Mobile and Ad Hoc Networking and Computing*, pages 147–148, 2000.

[59] A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of the 9th International Conference on Peer-to-Peer*, pages 99–100, Seattle, WA, September 2009.

[60] S.D. Nagowah. Aiding Social Interaction via a Mobile Peer to Peer Network. In *Proceedings of the* 4^{th} *International Conference on Digital Society*, pages 130–135. IARIA, 2010.

[61] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai. A Survey of Peer-to-peer Network Simulators. In *Proc. of the 7th Annual Postgraduate Symposium, Liverpool, UK*, volume 2, 2006.

[62] K. Nakano and S. Olariu. Randomized Leader Election Protocols for Ad-hoc Networks. In *Proceedings of the* 7^{th} *International Colloquium on Structural Information and Communication Complexity*, pages 253–267. Carleton Scientific, 2000.

[63] Ivan Ozhiganov. NFC Alternative: Transferring Data Between Mobile Devices Using Ultrasound. AZOFT, 2013. `http://rnd.azoft.com/mobile-app-transering-data-using-ultrasound/`.

[64] B. Kumar Panda, M. Das, B. Sahu, and R. Das. Impact of Mobility and Terrain Size on Performance of AODV and DSR in Mobile Ad-hoc Networks. In *Proceedings of the* 9^{th} *IEEE International Conference on Wireless and Optical Communications Networks*, pages 1–5, Sept 2012.

[65] C.E. Perkins and E.M. Royer. Ad-hoc On-Demand Distance Vector Routing. In *Proceedings of the* 2^{th} *IEEE International Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.

[66] G. P. Perrucci, F. H. P. Fitzek, and J. Widmer. Survey on Energy Consumption Entities on the Smartphone Platform. In *Proceedings of the* 73^{th} *IEEE International Conference on Vehicular Technology*, pages 1–6, 2011.

[67] X.T. Phan, Q.T. Minh, K. Nguen, N. Thoai, and S. Yamada. GICS: Group-Based Internet Connection Spreading Architecture for Disaster Recovery. In *Proceedings of* 29^{th} *IEEE International Conference on the Advanced Information Networking and Applications Workshops*, pages 478–483, March 2015.

[68] A. Pyattaev, K. Johnsson, S. Andreev, and Y. Koucheryavy. 3GPP LTE traffic offloading onto WiFi Direct. In *Proceedings of the IEEE International Conference on Wireless Communications and Networking*, pages 135–140, 2013.

[69] A. Pyattaev, K. Johnsson, S. Andreev, and Y. Koucheryavy. Proximity-Based Data Offloading via Network Assisted Device-to-Device Communications. In 77^{th} *IEEE Vehicular Technology Conference (VTC Spring)*, pages 1–5, June 2013.

[70] A. Pyattaev, K. Johnsson, A. Surak, R. Florea, S. Andreev, and Y. Koucheryavy. Network-assisted D2D communications: Implementing a technology prototype for cellular traffic offloading. In *IEEE Wireless Communications and Networking Conference (WCNC)*, pages 3266–3271, April 2014.

[71] V. Raychoudhury, J. Cao, and W. Wu. Top K-Leader Election in Wireless Ad Hoc Networks. In *Proceedings of the* 17^{th} *IEEE International Conference on Computer Communications and Networks*, pages 1–6, 2008.

[72] Ruckus. How interworking works: A detailed look at 802.11u and hotspot 2.0 mechanisms. Technical report, July 2013.

[73] S. Bernardo. Wi-Fi P2P Simulator. GitHub, 2013. `https://github.com/samuelbernardo/12_NearTweet2013.git`.

[74] S. Mangold. The Emulation Software to model 802.11 and other 802 Wireless Communication Systems, 2015. `https://github.com/schmist/Jemula802`.

[75] L. Santos and A. Ribeiro. My-Direct: A Middleware for P2P Mobile Social Networks. In *International Journal of Computer Networks and Communications*, volume 6, pages 177–196, 2014.

[76] Senthil Kumar. IoTivity Simulator, 2016. `https://wiki.iotivity.org/iotivity_tool_guide#simulator_eclipse_plug-in`.

[77] Z. Shijie, Z. Yanghong, and L. Jiaqing. Research on Simulators for Peer-to-peer Systems. In *Proc. of the 3rd International Conference on Computer Science and Network Technology*, pages 726–731, Oct 2013.

[78] I. Stojmenovic. Fog Computing: A Cloud to the Ground Support for Smart Things and Machine-to-Machine Networks. In *Proceedings of the IEEE Australasian Telecommunication Networks and Applications Conference*, pages 117–122, Nov 2014.

[79] T. Henderson. Communication Range in NS-2, 2011. `http://www.isi.edu/nsnam/ns/doc/node222.html`.

[80] Godfrey Tan, Allen Miu, John Guttag, and Hari Balakrishnan. *Blueware: Bluetooth Simulator for ns, MIT Technical Report*. `http://nms.lcs.mit.edu/projects/blueware/`.

[81] Inc. Thread Group. Thread technical overview. Technical report, Oct 2015. `http://www.threadgroup.org/resources/most-popular`.

[82] K. Townsend, R. Davidson, and C. Cufí. *Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking*. EBSCOhost ebooks online. O'Reilly, 2014.

[83] S. Trifunovic, B. Distl, D. Schatzmann, and F. Legendre. WiFi-Opp: Ad-hoc-less Opportunistic Networking. In *Proceedings of the 6th ACM Workshop on Challenged Networks*, pages 37–42. ACM, 2011.

[84] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. Jubert, M. Mazura, M. Harrison, M Eisenhauer, et al. Internet of things strategic research roadmap. In *Internet of Things: Global Technological and Societal Trends*, volume 1, pages 9–52, 2011.

[85] M. Villari, A. Celesti, M. Fazio, and A. Puliafito. AllJoyn Lambda: An architecture for the management of smart environments in IoT. In *International Conference on Smart Computing Workshops (SMARTCOMP Workshops)*, pages 9–14, Nov 2014.

[86] X. Y. Wang and P. H. Ho. Gossip-Enabled Stochastic Channel Negotiation for Cognitive Radio Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 10(11):1632–1645, Nov 2011.

[87] Y. Wang, A. V. Vasilakos, Q. Jin, and J. Ma. A Wi-Fi Direct Based P2P Application Prototype for Mobile Social Networking in Proximity (MSNP). In *Proceedings of the* 12^{th} *IEEE International Conference on Dependable, Autonomic and Secure Computing*, pages 283–288, 2014.

[88] H. Wirtz, T. Heer, R. Backhaus, and K. Wehrle. Establishing Mobile Ad-hoc Networks in 802.11 Infrastructure Mode. In *Proceedings of the 6th ACM Workshop on Challenged Networks*, pages 49–52. ACM, 2011.

[89] Hongyi Wu, Chunming Qiao, S. De, and O. Tonguz. Integrated cellular and ad hoc relaying systems: iCAR. In *IEEE Journal on Selected Areas in Communications*, volume 19, pages 2105–2115, Oct 2001.

[90] J. Zuo, Y. Wang, Q. Jin, and J. Ma. HYChat: A Hybrid Interactive Chat System for Mobile Social Networking in Proximity. In *IEEE International Conference on Smart City*, pages 471–477, Dec 2015.

[91] S. Zurbes. Considerations on link and system throughput of bluetooth networks. In *The 11th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, volume 2, pages 1315–1319, September 2000.