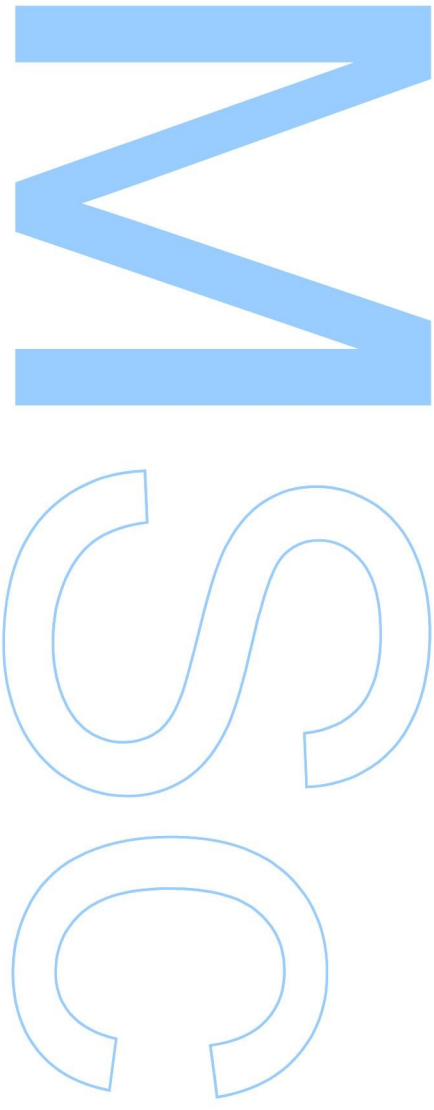
Simulation of mobile edge-cloud applications using Mininet-WiFi

Tiago Miguel Alves Castanheira
Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciências de Computadores
2018

Orientador
Eduardo Resende Brandão Marques, Professor Auxiliar,
Faculdade de Ciências da Universidade do Porto

Co-orientador
Luís Miguel Barros Lopes, Professor Associado,
Faculdade de Ciências da Universidade do Porto



Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Abstract

The increasing use of mobile devices in recent years led to the consideration of mobile edge-cloud applications, where nearby devices form ad-hoc networks using peer-to-peer communication. Research towards a general-purpose mobile edge-cloud middleware is being conducted by the Hyrax Project, along with a few case-study applications. Testing these applications in real-world with a large number of devices, however, is a very onerous task and virtually impossible to reproduce with fidelity.

As the main goal of this thesis, we described and developed a framework to simulate mobile edge-cloud applications under real world physical conditions, taking into account factors such as network setup, user mobility, and the use of peer-to-peer technologies. We extended the Mininet-WiFi platform, a lightweight emulator for mobile wireless networks, and used User-Generated Replays (UGR) Hyrax case-study as target to model our simulation. We performed simulations to study the benefit of exchanging connections for performance in file availability using WiFi, WiFi-Direct and cloudlets for multiple mobility models.

Keywords— WiFi, cloud, framework, simulation, mobility

Resumo

O uso crescente de dispositivos móveis nos últimos anos levou à consideração de aplicações para mobile edge-clouds, onde dispositivos próximos formam redes ad-hoc usando comunicação peer-to-peer. Neste momento já está a ser conduzida investigação para o desenvolvimento de um middleware para este novo tipo de paradigma, juntamente com algumas aplicações como caso de estudo. No entanto, testar estas aplicações no mundo real com um grande número de dispositivos é uma tarefa muito onerosa e virtualmente impossível de reproduzir com fiabilidade.

Com o principal objetivo desta tese em mente, foi descrita e desenvolvida uma framework para a simulação de aplicações para mobile edge-clouds sob condições físicas do mundo real, levando em consideração fatores como o setup da rede, a mobilidade do utilizador e o uso de tecnologias peer-to-peer. Foram feitas extensões para a plataforma Mininet-WiFi, um emulador para redes móveis sem fio e usámos como caso de estudo a aplicação do User-Generated Replay associado ao projecto Hyrax como alvo para a simulação. Realizamos simulações para estudar o benefício da troca de conexões para melhorar o desempenho na disponibilidade de ficheiros utilizando WiFi, WiFi-Direct e cloudlets para vários modelos de mobilidade.

Keywords— WiFi, cloud, framework, simulação, mobilidade

Acknowledgements

This work would not be possible without the help of everyone in my life, consequently I would like to give my thanks not just during my current work, but throughout my academic years.

I would to like to extend my deepest gratitude to both my supervisors, Professor Eduardo Marques and Professor Luís Lopes, for their tremendous effort in guiding me, for every piece of insightful information and all the hours spent helping with the development of this thesis.

Second to last, I would like to express my deepest appreciation to my friends for their friendship and companionship, for listening when needed, for all the help given and for all the nights out. I would like to thank in particular to Pedro Paredes, for his unwavering guidance throughout my academic years, to Gonçalo Pereira and Marco Lopes, for all the support and patience and last but not least, to those that were present in early years that for different circumstances I have part ways with.

Finally, I would like to express my gratitude to the pillars of my life, my family. I would like to thank them for their unstoppable support, for all the affection and love that paved the way for my next journey. This would not have been possible without the support and affection of both my brothers, André and Bruno Castanheira.

To my beloved mother

Contents

List of Tables

List of Figures

Acronyms

AP Access Point

D2D Device-to-Device

DCF Distributed Coordination Function

GO Group Owner

IP Internet Protocol

LAN Local Area Network

MEC Mobile Edge-Cloud

MLME Media Access Control Sublayer Management Entity

MS Mobile Station

P2P Peer-to-Peer

QoS Quality of Service

RSSI Received Signal Strength Indicator

SSID Service Set Identifier

SDN Software-Defined Network

TDLS Tunneled Direct Link Setup

UGR User-Generated Replays

WLAN Wireless Local Area Network

WPA WiFi Protected Access

Chapter 1

Introduction

The Internet has become a commodity for most of the world's citizens, it has given us the ability to connect to everyone and everything from where we stand. Connecting everyone using the Internet was only possible using our computers since mobile devices were not powerful enough to maintain a stable connection but in most recent years, with the advances made in wireless technology it is possible to connect everyone from everywhere with a panoply of mobile devices that we carry everyday, in the form of smartphones or tablets.

Mobile devices are now relatively powerful machines in terms of storage, memory, built-in sensors and networking capabilities. Moreover, they have become ubiquitous, a recent study shows that by 2018, 36% of the world population will own a smartphone [23]. Many mobile applications are traditionally deployed in conjunction with cloud systems, a paradigm known as mobile cloud computing. More recently, proximity-aware applications have driven the emergence of Mobile Edge-Cloud (MEC), where nearby devices can form a ad-hoc network using Peer-to-Peer (P2P) communication, possibly but not necessarily supported by thin-servers on the edge of the network (i.e. close to the users) known as cloudlets.

In the Hyrax project [1], research is being conduced on a general-purpose middleware for MEC development [18], along with a few of case-study MEC applications [19, 21, 22]. The special characteristics and aims of MEC have driven this research, such as: the integrated use of standard communication technologies like WiFi or 3G/4G with P2P technologies like WiFi-Direct, WiFi-TDLS or Bluetooth; the use of local computing / storage networking at the edge to reduce or remove dependence from traditional communications infrastructure and/or achieve lower network latency and congestion during operations; the challenges posed by the dynamics of device mobility and churn.

For instance, one of the case-study Hyrax applications, known as User-Generated Replays (UGR) illustrates these concerns [22]. Its main idea is to allow users to capture video replays at sporting events, and disseminate such videos to other users. Since the density of devices using a normal WiFi or 3G/4G infrastructure in a crowded place can congest this infrastructure, UGR employs P2P communications for direct video dissemination, combined more recently

with the use of cloudlets and mesh networking for video dissemination across distinct areas of a sporting venue. In this setting, devices can work as both servers and clients which presents the opportunity to engage in different scenarios. A number of aspects are interesting to model, such as the existing infrastructure for communications or the lack of it, user mobility patterns, the effect of device churn, in order to understand the limitations, potential, and scalability of different solutions.

1.1 Problem statement and Goals

Testing MEC applications, such as the Hyrax UGR discussed above, with a large number of devices in the real world is a very onerous task and virtually impossible to reproduce with fidelity. Aspects like mobility for a large numbers of devices would be impossible to reproduce due to the requirements (we would need a lot of users moving around) and the propagation of wireless signals in the real world are also very susceptible to interference from other sources.

The objective of this dissertation was to develop a framework to simulate how applications behave in MEC under real world physical conditions, taking into account factors such as user mobility, signal propagation models and the use of P2P technologies in addition to the nature of the applications at stake. As a supporting platform, we planned to use the Mininet-WiFi emulator [7], that makes it possible to account for these characteristics, coupled with a lightweight network virtualisation scheme and the use of unmodified program binaries.

The aim was also to employ the intended framework to analyze a few case-study applications that have been considered in the scope of Hyrax, like UGR and possibly others. The intent was to shed light on the interplay between factors such as the scale of the system, mobility, states of the system, and different parameterizations of an application or the network setup.

1.2 Contributions

With the thesis goals in mind, our contributions are as follows:

- Give an overall description of simulators, emulators and testbeds followed by an evaluation of the set of frameworks already available to perform network simulation;
- A framework to simulate how applications behave in MEC under real world physical conditions using Mininet-WiFi;
- Simulation of diverse network scenarios, based on an existing case-study to evaluate performance of such applications;
- Results based on the simulation, where we evaluate the performance of existing technologies such as WiFi-Direct, WiFi and the use of cloudlets.

1.3 Thesis Structure

The remainder of this thesis is structured as follows.

Chapters 2 and 3 are introductory for our work. Chapter 2 discusses the state-of-the-art for network simulation approaches, mobility patterns, signal propagation models, and other related work. Chapter 3 introduces a background of the technologies present, explains the case-study that was used as our base model for the implementation and adds insight to Mininet-WiFi emulator.

Chapter 4 and 5 concern the simulation framework. Chapter 4 describes the design and implementation of our simulation framework, by identifying the simulation requirements we had in mind and then describing the main aspects of implementation. Chapter 5 provides an evaluation of results we obtained using the simulation framework.

Finally, Chapter 6 provides concluding remarks and discusses directions for future work.

Chapter 2

State of the Art

In this chapter, as our work is to simulate different Mobile Edge-Cloud (MEC) scenarios, we must study which simulation frameworks are powerful enough for this task. First, we describe why we must simulate. Second, we must understand which frameworks can be used and which models will best adequate our simulation scenarios, as a result we describe some simulation frameworks as well as a brief resume of some mobility and propagation models. We then do a comparison and evaluate what best fits our requisites.

2.1 Experimental wireless network simulation

Since the beginning of first networks, network frameworks have been designed to provide a better testing environment to model network behavior as performing the real implementation and testing of such systems in the real world would become a very expensive task. Several tools for performance evaluation have been mentioned in the literature [13], where each one has its own benefits. We must find a clear insight into which ones are better suited to deal with the challenges that are going to be presented in this thesis.

2.1.1 Simulators, Emulators and Testbeds

There are three types of frameworks that are used in experimental research to evaluate performance in networks which can be useful to evaluate wireless applications as well: Simulators, Emulators and Testbeds.

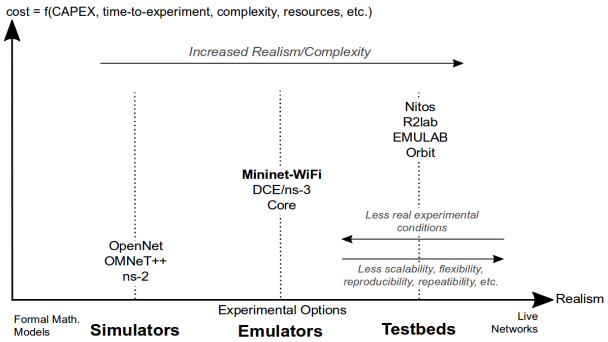
- Simulators are software-based frameworks that model real environment, they are popular and effective approach to design networks. They attempt to accurately model and predict the behavior in different scenarios.
- Emulators are different from Simulators as they can run the same code on real platforms, they are an hybrid approach where the combination of software and hardware are used to

provide for the different aspects of the simulation. Some aspects like traffic and links are simulated using software and others are implemented on real hardware.

- TestBeds are a physical infrastructure that supports experimental research with real-world settings by placing nodes in a control environment, they are a gap between both Simulators and Emulators, they provide for an environment for protocol testing and evaluation that is similar to an actual deployment.

Characteristic	Simulators	Emulators	Testbeds
Total Cost	●○○	●○○	●●●
Overall Fidelity	●○○	●●○	●●●
Replay Real Traces	●●○	●●○	●●●
Real Applications	●○○	●●●	●●●
Traffic Realism	●○○	●●●	●●●
Timing Realism	●●●	●●○	●●●
Scalability	●●●	●●○	●○○
Maintainability	●●●	●●●	●○○
Flexibility	●●●	●●●	●○○
Replication	●●●	●●●	●○○
Isolation	●●●	●●○	●●●

(a) Simulators, Emulators and Testbeds Strengths (Originally from [27] adapted in [6])



(b) Trade-offs of different wireless experimental platforms [6]

Figure 2.1: Simulators, Emulators and Testbeds Strengths and Trade-offs of different wireless experimental platforms

Each one has their own advantages and disadvantages such as scalability, fidelity or repeatability. In Figure 2.1a there is a summarized description of the characteristics common to all frameworks with detailed evaluation to each one, so we can better understand what type of approach for one's research is the most relevant. Also, in Figure 2.1b we can observe the trade-offs of using each one of the frameworks.

2.2 Testbeds

Testbeds, as mentioned before, are an environment to evaluate network performance of algorithms and protocols before implementing these systems in the real world and, although they are a good approach, they do incur in a high maintenance cost. There are a couple of testbeds that can be suited towards the simulation of MEC, respectively R2Lab [25], Nitos [15] and Orbit [16]. R2Lab is a testbed primarily for reproducible research in wireless WiFi and 4G/5G networks. It is located in a anechoic chamber and provides several wireless devices fully customizable with access to software tools to support easier experimentation. Nitos is another testbed that also focuses on supporting experimental-based research in the area of wired and wireless networks. It is comprised of three different scenario deployments, one outdoors, one indoors and an office testbed. At last, we have Orbit, it is an emulator and field trial network testbed designed to achieve reproducible experimentation with support for protocols and application evaluation. Its

laboratory uses a novel approach that involves a large two dimensional grid with many 802.11 radio nodes that can be dynamically connected into specified topologies.

2.3 Simulators and Emulators

In this section we describe and evaluate five simulation frameworks which best accommodate our needs in order to decide which is the most suited for our work. The following simulation frameworks are WSNet, NS-3, OMNET++, OpenNet and Mininet-WiFi.

2.3.1 WSNet

WSNet [8] is an event driven simulator for wireless sensor networks, it is capable of simulating nodes with different energy sources, mobility, interference models, applications and routing protocols. Its architecture is presented as blocks that model the characteristics and properties of the different simulation aspects such as radio medium and sensor nodes. When simulating, for example, radio propagation, interference or modulation are done in separated blocks. Another property of WSNet is the ability to simulate environmental aspects like fire, extending it towards its physical characteristics (e.g, temperature) that are visible in the nodes system. This can result of the node's death, being useful for disaster scenario simulation. WSNet goal is to offer scalability, extensibility and modularity for the integration of new protocols, it supports some 802.15 and 802.11 Distributed Coordination Function (DCF) IEEE standards. It is written in C, hasn't got an update since 2009 and although it has incorporated mobility it does not follow any traditional model.

2.3.2 NS-3

NS-3 [10] is a discrete-event simulator mainly used for research and educational studies. It functions by keeping track of the events in a queue that are scheduled to execute at specified simulation time. The simulator is capable of simulating wireless networks like Wi-Fi, LTE, other wired connections and even non-IP based networks. It supports the various 802.11 Wi-Fi physical layers, a variety of mobility models, propagation models and routing protocols, either static or dynamic such as OLSR and AODV. Even thought it is a discrete-event simulator it supports a real time scheduler for "simulation-in-the-loop" use cases. Also, there have been developed frameworks to help running unmodified applications of the entire Linux kernel networking stack within ns-3, so we can use, for example, Linux ping instead of the ns-3 implementation of it. It still is widely used and maintained, receiving updates regularly. It is written in C++ and Python. NS-3 does not have Wi-Fi scan mechanisms not allowing layer-2 handover of mobile nodes for different channels.

2.3.3 OMNeT++

The OMNeT++ [26] is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. It is not a simulator *per se*, but a environment to build simulators. OMNeT++ was extended with many routing protocols for mobile ad-hoc network by the INET framework, which also allows for simulation of the Internet stack, routing protocols, support for mobility and more. This means, OMNeT++ is capable of simulating IEEE 802.11 protocols and modeling of wired and wireless communication networks. OMNeT++ is very active, receiving constants updates. Since OMNeT++ is a modular simulation library there have been developed many modules from different creators which makes it harder to work with combined modules. Also, OMNeT++ has it own implementation of switch/AP and does not support third-party controllers.

2.3.4 OpenNet

OpenNet [4] is a simulator for Software Defined WLAN built on a combination of both NS-3 and Mininet. The main idea behind this simulator is to take advantage of both simulators, since they have a higher fidelity for the development of new protocols. It uses the link layer of mininet with the physical layer of ns-3 linking them together by creating a "Tap Device", which is used to simulate the behavior of mobile nodes or access points depending on the link layer module it uses. Then, mobile nodes connect with each other using an emulated channel provided by ns-3. All the mobility models, propagation models and routing protocols of ns-3 are present in OpenNet. Since ns-3 does not provide a Wi-Fi scan mechanism, some modifications were made to ns-3 in conformity to IEEE 802.11 specification as a solution to provide handover in different channels.

2.3.5 Mininet-WiFi

Mininet-Wifi [7] is a lightweight emulator based on the mininet project. Mininet is described as software for rapid prototyping for Software-Defined Network (SDN), its main goal is to emulate a complete network of hosts, links, and switches on a single machine [24]. Mininet-WiFi is an addition of classes on top of mininet, supporting all the normal SDN emulation capabilities and virtualization of both WiFi stations and access points using standard Linux wireless drivers. From these additions, it supports mobility and propagation models.

The WiFi virtualization is possible by using Linux drivers, namely the mac802.11_hsim and namespaces. The drivers allow for the simulation of the 802.11 IEEE radio standards and namespaces are used for the creation of the virtual networking, providing for a set of interfaces, Internet Protocol (IP) addresses and routing tables. Mininet-WiFi is updated regularly, it is written in Python and uses open-source software.

2.4 Comparison

Given the overview of the simulation frameworks it is important to distinguish between them. In order to do this, we establish metrics that can help us create a distinction so we best evaluate which will be better suited for our research. Since the author in [6] already has given a good evaluation, we use some of his metrics and add on top of them.

The following metrics that we have chosen are:

- Type - This entry is to define if it is an Emulator or Simulator.
- Source Type - If it is Open Source or Proprietary.
- Programming Languages - In what programming languages it has been written.
- Supported Protocols - What protocols it supports, respectively 802.11 IEEE standards.
- Mobility Support - If it supports mobility models.
- Propagation Support - If it supports propagation models.
- Latest Activity - When was the year of the last update.

Table 2.1: Comparison between simulation frameworks (Adapted from [6])

Software	Type	Source Type	Programing Languages	Supported Protocols	Mobility Support	Propagation Support	Latest Activity
WSNet	Simulator	Open	C	IEEE 802.15, 802.11 DCF	Yes	Yes	2009
NS-3	Emulator	Open	C++/ Python	IEEE 802.11, LTE	Yes	Yes	2017
OMNeT++	Simulator	Open	C++	IEEE 802.11	Yes	Yes	2017
OpenNet	Simluator/ Emulator	Open	C++/ Python	IEEE 802.11, LTE	Yes	Yes	2017
Mininet-WiFi	Emulator	Open	Python	Any (L3 - L7), IEEE 802.11, 802.3	Yes	Yes	2017

In Table 2.1 there is a resume of what features each framework supports. This information was obtained from the main articles of each framework. As we can see, most of the simulation tools have identical features so we must choose from them based on our work and prior research of such frameworks. It is important to notice that since our objective is to be able to use software towards the simulation of the MEC we left out testbeds from the analysis.

From the analysis and description of the frameworks, we have chosen Mininet-WiFi. Mininet-WiFi would be the best approach towards our research, Mininet-WiFi is an emulator that uses Linux drivers for the simulation, mostly mac80211_hwsim for radio simulation, it allows for the use of real code without kernel modifications or application code alteration, it supports IEEE 802.11 standards from open-source software, it includes support for WiFi-Direct and WiFi Mesh and has an easier support for the addition of frameworks to best work with SDN. We concluded

that all of our needs are satisfied under Mininet-WiFi since it has all the utilities that we need to simulate Hyrax applications in MEC scenarios.

The other frameworks were eliminated based on the simplicity of Mininet-WiFi and based on some specific limitations of each framework. As an example, WSNet mobility does not follow any particular model or for example, NS-3 can provide the same environment as Mininet-WiFi but it needs frameworks to fulfill such requirements.

2.5 Mobility Patterns

Since mobility is one of bigger limitations towards having a stable connection, in this section we survey several mobility models that have been published for mobile ad hoc networks for performance evaluation of various protocols. We divided the mobility models in two groups, one where we focus the behavior of single entities and other where we focus on group mobility.

2.5.1 Single Entity Mobility Models

The following models are used for single entities, where the model does not take into account the fact that other nodes are present in the simulation.

2.5.1.1 Random Walk

The Random Walk mobility model was developed to simulate a completely random movement resulting in unpredictable paths, based on a Stochastic Walk. In this mobility model, a mobile node moves from its current location to a new location by randomly choosing a direction and speed in which to travel [3]. Furthermore, speed and direction are fully random, possibly defined within a range, not accounting for previous speeds and locations. Thus, the mobility model generates unrealistic movements such as full stops following by sharp turns. In Tracy Camp et al. [3] they concluded that if the direction is given a small number and traveled distance is short, the Random Walk mobility model can be used to evaluate semi-static networks as the nodes do not roam far from its original position.

2.5.1.2 Random Waypoint

The Random Waypoint mobility model follows the same concept of the Random Walk, differing only before changing direction and speed, where the mobile nodes have paused times. Instead of being always moving, the mobile nodes stay still for a specific time, then it chooses a location and sets its direction and speed going towards that location. When it reaches that "Waypoint", it pauses again for a period of time, after resuming this process again. Following this mobility model, Tracy Camp et al. [3] described a limitation towards the initial number of neighbors,

where it has a higher percentage in the initial seconds. They propose various solutions to this as simple as ignoring the first seconds of the simulation and letting the mobility pattern stabilize. In addition, when using the Random Waypoint mobility model one must account to the various aspects of the simulation variables, like speed and paused time. In a situation where the mobile nodes stay still for a long period of time and move very little with fast speed, the network's topology hardly changes, which means the network is most stable when there is no mobility.

2.5.1.3 Random Direction

Random Direction mobility model was developed to correct the problem with the Random Waypoint mobility model. In the Random Waypoint model, the number of neighbors follows a density wave pattern (converging in an area and then leaving it). The mobile nodes following Random Waypoint choose a path that either goes to the middle since its where there is the most available destinations to choose from or reaches a destination that is only achievable traveling through the middle of the simulation. The Random Direction, in order to solve this problem, tries to provide for a constant number of neighbors. In this model, the mobile node travel towards the edge of the simulation. When it reaches it, it chooses a different angular position for the direction followed by changing its speed and then continues the process.

2.5.2 Group Mobility Models

In this case, the following models are used for group mobility, where the nodes behave as groups. The mobility models described are the Reference Point Group and the Time Variant Community.

2.5.2.1 Reference Point Group Mobility Model

This mobility model was first described in [11], where its main focus is to simulate group behavior. It creates groups which have a logical center, this logical center is then what defines the groups motion behavior controlling aspects such as location, speed or direction. This mobility model defines the motion of each group and the motion of each mobile node in each group. The RPGM model simulates its path according to the group logical center defining a sequence of check points to given time intervals, where each group moves from one check point to another. All of this is done by creating a group motion vector that is computed at each check point, as well as a random motion vector to represent the motion of each mobile node.

2.5.2.2 Time Variant Community Mobility Model

The Time Variant Community Mobility model was based on real wireless Local Area Network (LAN) user traces to mimic realistic mobility characteristics observed from daily lives. The

motivation behind this model was what the author in [12] describes as *skewed locations*, location where the mobile nodes spend most of its time and *periodical re-appearance* at the same location.

This model is divided into two types of periods of time: Normal Movement Periods and Concentration Movement Periods. It starts with a normal movement period where each time period it is assigned a community to each node. Communities is an abstraction of a location where mobile nodes spend most of its time. For each time period there are two modes of movement, local epoch and roaming epoch. In a local epoch, the node mobility is confined to its community and in a roaming epoch the node is free to move in the whole simulation area[12]. Also, different nodes pick different communities as their main locations, causing a dynamic mobility pattern. These different time periods with different movement periods are designed so there is a higher probability of a node visiting their communities, resulting in a periodical reappearance of a node in the same location.

2.6 Propagation Models

Propagation models are models that calculate the power of the signal received by the stations, known as Received Signal Strength Indicator (RSSI). These models follow a series of mathematical equations that calculate several signal aspects and translate them into equivalent network attributes, like the expected packet loss, the range at which there are communications or interference from the different sources. The RSSI value is computed taking into account the transmission power, antenna gains of transmitter and receiver and Path Loss. Then with these values it is possible to calculate the maximum supported rate during communication between two nodes. Equation 2.2 shows how Mininet-WiFi calculates its RSSI, where the propagation models are used to calculate the Path Loss.

$$PathLoss = PropagationModelFormula \tag{2.1}$$

$$signalStrength = pT + gT + gR - PathLoss \tag{2.2}$$

In order to use a better suited propagation model, we describe some that can be useful when simulating the various MECs scenarios.

- Free-Space Propagation Model - It was first described in [9], it follows the Friis propagation loss model equation, it is valid only for propagation in free space within the so-called far field region, this meaning it assumes there is no obstacles or terrain irregularities. Since Friis' model behaves as there are no obstacles it rarely simulates reality.
- The Log-Distance Propagation Model is an indoor or largely populated areas model, which as the name says, calculates the Path Loss over distance between transmitter and receiver based on a logarithmic metric.
- International Telecommunication Union (ITU) Propagation Model is a model that estimates the Path Loss in closed areas such as inside a room. It is more suitable for indoor use as it takes into account the number of floors between receiver and transmitter.

- The Two-Ray-Ground Propagation Model is an outdoor model, where the original equation comes from Rappaport's book. It predicts the path loss between a transmitting antenna and a receiving antenna with both the line of sight component and the multipath component formed by the ground reflective wave.

2.7 Related Work

In order to understand which scenarios and topologies are used in the MEC we must study other implementations of similar work, this gives us a better understanding of which metrics we can evaluate and challenges that derive from these scenarios, followed by the solutions that each author presented.

2.7.1 Using Edge-Clouds to Reduce Load on Traditional WiFi Infrastructures and Improve Quality of Experience

In Pedro Silva et al. [22], MECs utility was extended to traditional WiFi infrastructures to improve quality of experience, the authors were able to offload the network using as case study an Android application for soccer replays, where 56% of all downloads were made at the edge and the speed of downloads was 3 times higher. Furthermore, it was concluded that the use of MEC can outperform in both power management and computation power the traditional cloud infrastructure only if independent datasets are used, as other types of data increase the number of interactions causing a higher impact, such as reducing battery performance.

Understandably, since the users of the application are in a room watching a soccer game, there is barely no mobility, but it presents an overview of how the new MEC paradigm can increase the throughput versus a normal WiFi scenario.

2.7.2 Simulation of Algorithms for Mobile Ad-Hoc Networks

In this thesis, Joaquim Silva [20] provided us with a test for different overlays of MECs scenarios in a game stadium. The idea was to reduce the work on the server, using an application for replays just like in [22] where users can download previous plays in their smartphones, where metrics like churn with different probability and scalability were evaluated. The overlays that were implemented were Ring Overlay, Star, Peer-to-Peer and Scatternet. In the case of Ring Overlay, most files could be downloaded from the MEC, around 80% to 90%, when the network had a size of 64 nodes, but with the increase of nodes to bigger values, such as 128, these values were lower due to the increasing number of hops to find a file locally, this result coming from the fact that to find a file locally, a token was sent to every node saturating the medium and reducing bandwidth. The Star overlay was a good approach, being a scalable one, but it had a single point of failure in the master node which was responsible for providing the connection

stability. The number of hops in this case was only one or it used the server. Although the results, the author referred to this as unfeasible due to current limitation of technologies on smartphones. The Peer-to-Peer overlay was set to test different hops and showed that allowing an high number of hops to search for the files to be beneficial only on small network because it is not scalable. Also, increasing the number of hops resulted in flooding the network which is never a good solution. The last overlay tested was the Scatternet, this implementation showed that there was no loss of performance as the network grows. The percentage of file downloaded locally was higher when "Multi Group Search" was allowed, the groups formed could download from other groups, but it also increased the number of hops to find files.

Regarding churn, most of the overlays presented the following: the higher the churn the least downloads made locally and in total. The only situation where churn was better was when it reduced the number of nodes in the network, it made the look-up of files faster since the number of hops was smaller. It is also important to mention that when the churn effect is higher, in order to have a fault tolerant and churn resilient network, having a less structured overlay approach should be used. Lastly, when evaluating all the best solutions, the author referred to an hybrid solution could possible be the best approach, but in terms of delay, amount of messages exchanged and reduction of server access using a star/scatternet would result in a better performance versus the other overlays. So when the game is playing, a structured overlay should be used, since users are sitting down and at mid game, when there are intermissions, a less structured approach might be better.

2.7.3 Hybrid Physical-Virtual Software-Defined Wireless Networking Research

The concept of MECs with SDN was done in [5]. In this use case experiment, Ramon and Rothenberg were able to connect real users to a virtualized infrastructure with their smartphones using normal WiFi. They were able to interact with nodes forming a subnetwork mesh and could access the global Internet after having its traffic processed through a controller flow. This was possible using Mininet-WiFi, showing the ability of integrating physical and virtual environments, AP-management using a controller providing some Quality of Service (QoS) control through metric rules and emulation of wireless mesh network with basic routing capabilities.

2.7.4 Time-Aware Publish/Subscribe for Networks of Mobile Devices

João Silva et al. in [19] presents a new time-aware publish/subscribe architecture for mobile devices with two different materializations, one based on a simplistic flooding approach and another based on a geographical approach that follows a data-centric storage using a geographical hash table as storage subtract. The idea behind THYME is to focus on data dissemination between mobile devices. In order to evaluate THYME they use simulation, where they use the NS-3 emulator. The authors use mobility, but only stationary nodes render routing information or form the geographical hash table, if they move they stop forwarding messages (they still process

receiving beacons) and when they stop they resume the protocol. When moving the nodes follow the Random Waypoint mobility model with some alterations in the moving-stopping metrics, with three speeds: 0.6 m/s, 1.4 m/s and 2.5 m/s. It was concluded that, with the flooding approach, churn was problematic since the publications were done locally and the data-centric storage with the geographical hash table would be a better approach but with low mobility.

Chapter 3

Background

In this chapter, we describe the core background of our work. It starts by describing a case-study MEC application from the Hyrax project, the User-Generated Replays (UGR), that inspired us during the identification of necessary features in the simulation framework and the experiments we conducted afterwards (Section 3.1). Next, we provide an explanation of the main aspects of WiFi-Direct, the peer-to-peer communication technology considered in the thesis (Section 3.2). Lastly, we give a broader description of the architecture of Mininet-WiFi, the base platform for our simulation framework (Section 3.3).

3.1 User Generated-Replays case study

The UGR is a major case-study application of the Hyrax project [17, 22], providing a good motivation for the validation of Mobile Edge-Cloud (MEC) applications through simulation. The UGR application allows users to capture and share videos in a crowded venue, e.g., during a sport event. Traditional applications rely on infrastructural communications for data dissemination,

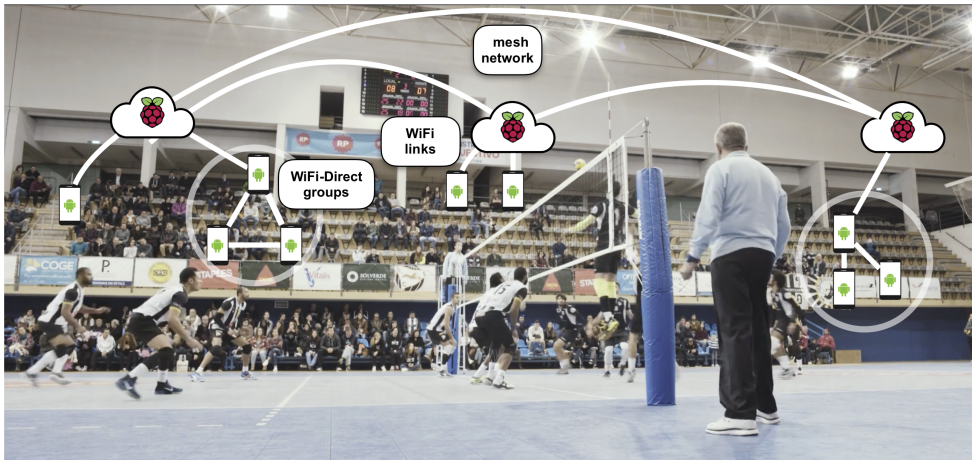


Figure 3.1: Field experiment for the UGR scenario (picture from [17]).

leading to traffic congestion, e.g., infrastructural WiFi and 3G/4G access can often be quite slow in crowded venues. UGR mitigates this problem by leveraging edge-cloud networks formed by nearby devices and/or proximity cloudlets, using infrastructural communications as a last resort.

Figure 3.1 depicts a photo of a UGR field experiment we conducted recently in the real-world setting of a Portuguese volleyball league game [17]. The picture also shows a schematic illustration of a hybrid edge-cloud architecture supporting the UGR functionality during the experiment. Lightweight cloudlets enabled through Raspberry Pi devices were deployed at different locations in the game venue. Each cloudlet served as WiFi access point and hosted a video storage service, allowing devices to upload and download videos from it. Cloudlets were also connected among themselves through a mesh network, such that videos uploaded by devices to a particular cloudlet would be automatically disseminated to the remaining cloudlets through the mesh network, thus allowing users at different locations to share videos almost in real-time. Meanwhile, mobile devices could spontaneously form WiFi-Direct groups, that would act as video caches, i.e., download requests for videos available in a group would be served locally bypassing the cloudlet layer.

3.2 WiFi-Direct

The WiFi-Direct standard was released in 2010 by the WiFi Alliance consortium. Its main objective is to allow the creation of network groups, creating local wireless networks without the use of any physical infrastructure. The standard is commonly supported by iOS and Android devices. Using WiFi-Direct The devices can discover each other and arrange in a very specific way automatically. When a connection is made, one of the devices is set as a Group Owner (GO). The GO acts as a sort of Access Point (AP) responsible for the routing of packets between the group members. Any device can become GO or a group member depending on the negotiation algorithm.

There are many different approaches to negotiate how to group is created. The most standard one results from the use of WiFi scan mechanisms in which a device looks up groups already created. After this scanning mechanism takes place, WiFi-Direct takes advantage of discovery algorithms to find other peer-to-peer devices. As two devices encounter each other they enter a negotiation phase. This negotiation, called Go Negotiation phase, results from a three-way handshake where they establish each other roles and are able to start communicating. The GO is required to support a DHCP server to provide IP addresses and is the only device that can connect to an external network.

Anther approach, the one used in this work, is to use the legacy mode that WiFi-Direct provides. It works by manually initializing a device as a GO, broadcasting a network with a Service Set Identifier (SSID), becoming what is essentially known as “soft AP”. It is called “legacy” mode, because the members need not formally be WiFi-Direct group member, but instead connect to the WiFi-Direct GO the same way as to a standard WiFi AP.

3.3 Mininet-WiFi architecture

.

Mininet-WiFi [7] is an extension of Mininet [14] for mobile wireless networks. Mininet supports the creation of virtual networks with an arbitrary number of hosts within a single Linux host. Through a lightweight virtualization scheme, Mininet provides the illusion of multiple hosts that may run unmodified Linux binaries/utilities and use the standard Linux networking stack. For scripting, the Mininet functionality is exposed through a Python API. Building upon Mininet, Mininet-WiFi adds support for various types of WiFi communications (standard WiFi, WiFi-Direct, mesh networking, etc) and different types of wireless nodes (mobile stations, static hosts, or access points). Moreover, it provides support for the definition of mobility patterns and emulation of the wireless physical medium and associated signal propagation models. A Mininet-WiFi screenshot is shown in Figure 5.1, illustrating a particular simulation with access points and mobile stations deployed over a certain physical area.

Mininet-WiFi can be divided into two parts, the kernel-space and the user-space, with the architecture illustrated in Figure 3.2.

In kernel-space we have the `mac80211_hwsim`, a Linux kernel module for simulation of IEEE 802.11 radios for `mac80211`. This Linux kernel module was designed to match very closely with real world Wireless Local Area Network (WLAN) hardware [2], it is responsible for creating virtual WiFi interfaces. Also in kernel-space we have Media Access Control Sublayer Management Entity (MLME), provided by the `mac80211`, which is a management entity for where the physical layer MAC state machines reside allowing for authentication, association and others.

On user-space we have utilities like `hostapd`, `wpa_supplicant`, `iw`, `iwconfg`, that are available from the Linux kernel module. `Hostapd` is a daemon responsible for MLME for access points in user-space. Utilities `iw` or `iwconfig` are used for configuration of interfaces and getting information from wireless interfaces. In addition, `wpa_supplicant` is used in conjunction with `hostapd` for

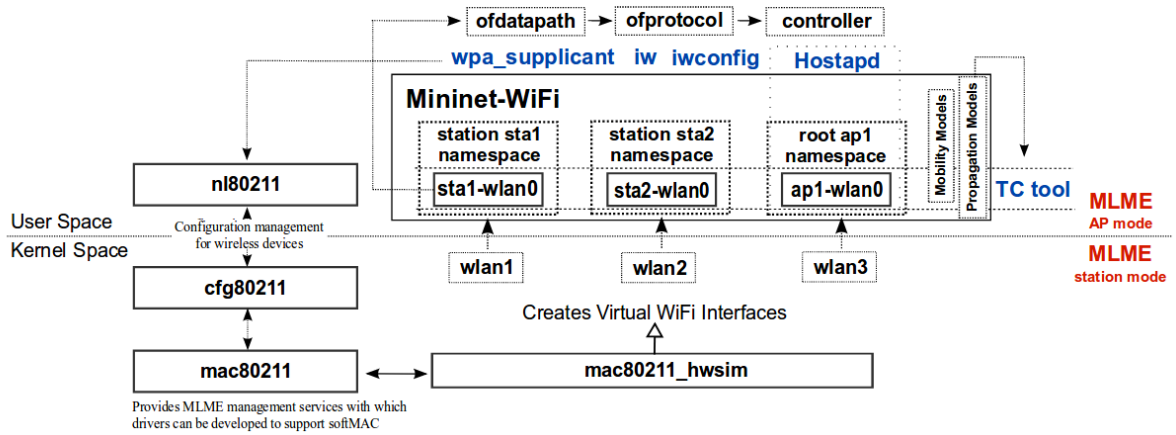


Figure 3.2: Mininet-WiFi Architecture [6]

support of WiFi Protected Access (WPA). The module `wpa_supplicant` is also useful to provide support for establishing Device-to-Device (D2D) communication simulating WiFi-Direct, this is done passing commands to the `wpa_cli` which is a text-based frontend program for interacting with `wpa_supplicant`.

Another core functionality is the Linux `Traffic Control`, used in the user-space, this utility program is used to configure the Linux kernel packet scheduler, which can be used to manipulate delays, rate of transmission, latency or loss to simulate real world network connections. An alternative for the Linux Traffic Control, that can also be used, is the wireless medium simulation tool, `wmediumd`.

`Wmediumd` provides for a more accurate simulation, providing simulation of frame loss and delay for wireless nodes. Since virtualization of radios use the same virtual medium for all of the wireless nodes internally `wmediumd` is as a solution to provide for full isolation from each other.

Communication between access points and stations to the wireless device driver use the `cfg80211`, the configuration API from the Linux kernel module that communicates to `mac80211`. From that, mac80211 then communicates directly with the WiFi device driver through `nl80211`, which is a netlink socket, that communicates between user-space and kernel-space.

Mininet-WiFi is able to abstract each station and access point using Linux OS `namespaces`, a feature that allows that one set of processes sees one set of resources while another set of processes sees a different set of resources.

Chapter 4

Design and implementation

In this chapter, we present the main aspects of design and implementation of our simulation framework. We first present the simulation requirements we had in mind. (Section 4.1). In correspondence to the requirements, we explain the extensions and adaptations of Mininet-WiFi to fulfill part of them (Section 4.2). and the design and implementation of a mobile file synchronization service inspired by the UGR application that runs embedded in all hosts of the Mininet-WiFi environment (Section 4.3).

4.1 Requirements

Our requirements for the simulation framework were two-fold:

1. To extend Mininet-WiFi to accommodate for WiFi-Direct group formation and cloudlets.
    - WiFi-Direct support: within Mininet-WiFi, the main simulation cycle essentially accounts for WiFi stations (moving in space) and APs (with fixed location). We should make it possible that nearby mobile devices are able to form WiFi-Direct groups, rather than just letting stations connect to APs.
    - Cloudlet support: also support the concept of cloudlets, that take the form as fixed-location hosts, possibly inter-connected by a mesh network. Each cloudlet should also be attached / associated to a WiFi AP and be able to run arbitrary programs.
2. To implement a mobile file synchronization service that would be able to model file dissemination in edge clouds. In other words, this is the target application we wished to simulate, supported by Mininet-WiFi.
    - Each station or cloudlet peer in the Mininet-WiFi should be able to run the service.
    - The service should allow each peer to act as producer and consumer of file content, seizing upon WiFi or WiFi-Direct connections established over time with other peers.

4.2 Mininet-WiFi extensions

4.2.1 Methodology

In order to extend Mininet-WiFi we needed to patch functionality in the base code. For this purpose, we made use of Python's method overriding facilities. allowing our changes to be well identified and incrementally defined in external manner to Mininet-WiFi's in a single file with no more than 300 lines of Python code. This contrasts with what would be a more direct approach, i.e., changing the base source code, that would scatter the changes across several spots, and be more prone to error in terms of maintenance.

The approach is illustrated by a short fragment of our code in Figure 4.1. In the fragment, method `configureLinks` in the `mobility` class of Mininet-WiFi is replaced by `myConfigureLinks`. The line of code in the fragment that performs the change on-the-fly is
`mobility.configureLinks = myConfigureLinks`.

The change happens in the context of `EdgeCloud`, a Python class that extends the `Mininet_WiFi` class, defining the core API of Mininet-WiFi. Simulation scripts that use `EdgeCloud` in place of `Mininet_WiFi` can thus thus use the augmented simulation functionality. However, as in the case of `myConfigureLinks`, there is a need to "patch" methods in other Mininet-WiFi classes, it is not only a case of extension, hence the constructor replaces the functionality of a few other methods within the base code.

```
from mininet.WiFi.mobility import mobility
...
@classmethod
def myConfigureLinks(cls, nodes):
  for node in nodes:
    if node.name.startswith('staHost'):
        continue
    cls.check_association(node, 0, ap_wlan=0)
    if isinstance(node, Station):
      if EdgeCloud.WiFidirect == 1:
        EdgeCloud.controlStateWiFiDirect(cls, node)
    eval(cls.continue_params)
...
class EdgeCloud(Mininet_WiFi):
   ...
   def __init__(self, root_dir, seed, WiFidirect, **kwargs):
      ...
      mobility.configureLinks = myConfigureLinks
```

Figure 4.1: Scheme for changes to Mininet-WiFi.

4.2.2 WiFi-Direct support

To enable WiFi-Direct with minimal changes, we resorted to supporting WiFi-Direct in so-called "legacy mode", whereby a mobile station can essentially become a WiFi-Direct group owner but be seen as an AP to which other mobile stations can connect via traditional WiFi means. A mobile station becomes a WiFi-Direct group owner at instants dictated by a Poisson random process with configurable frequency, having as pre-conditions that the station is not already a group owner and that it is not connected to another station via WiFi-Direct. In reverse, WiFi-Direct is dismantled if the group stays empty (has no connections to it) for a certain duration.

```
@classmethod
def controlStateWiFiDirect(cls, mobilitycls, sta):
  if ... :
     ...
     EdgeCloud.enableWiFiDirectGO(sta, mobilitycls)
     ...
  if ... :
     ...
     EdgeCloud.disableWiFiDirect(sta, mobilitycls)

@classmethod
def enableWiFiDirectGO(self, sta, cls):
  ... create WiFi-Direct configuration for wpa_supplicant and start it ...
  cls.aps.append(sta)

@classmethod
def disableWiFiDirect(self, sta, cls):
  ... stop wpa_supplicant ...
  cls.aps.remove(sta)
```

Figure 4.2: Code sketch for WiFi-Direct handling.

.

The use of the "legacy" mode is helpful in implementation terms regarding WiFi-Direct activation/deactivation, given that Mininet-WiFi works by maintaining a list of APs and a list of stations to handle aspects such as mobility or connection handover. WiFi-Direct group creation/destruction correspond respectively to adding/removing mobile stations to/from the list of APs maintained by WiFi-Direct. The other necessary operations relate to the appropriate activation/deactivation of `wpa_supplicant`. Figure 4.2 provides a simple sketch of the three methods in `EdgeCloud` that support WiFi-Direct group handling. The `controlStateWiFiDirect` method is invoked for every station in every simulation step, and may trigger WiFi-Direct group creation or destruction, respectively implemented by `enableWiFiDirectGO` and `disableWiFiDirect`.

A sample `wpa_supplicant` configuration files, generated on-the-fly by `enableWiFiDirectGO` (discussed above), is shown in Figure 4.3. The file relates to aspects such as the network SSID name, WiFi-Direct setup, and security scheme. After the `wpa_supplicant` daemon is launched,

the auxiliary command `wpa_cli -i staName-wlan01 p2p_group_add freq=2412 ...` is invoked, indicating `wpa_supplicant` to define a WiFi-Direct group owner with a WiFi frequency of 2412 Hz. The frequency is the same as for APs, allowing WiFi-Direct stations to connect to APs and have group members at the same time.

```
ctrl_interface=/var/run/wpa_supplicant
network={
    mode=3
    disabled=2
    ssid="DIRECT-GO-sta26"
    key_mgmt=WPA-PSK
    proto=RSN
    pairwise=CCMP
    psk="mysecret"
}
ap_scan=1
p2p_no_group_iface=1
p2p_go_ht40=0
device_name=sta26
device_type=1-0050F204-1
p2p_go_intent=15
p2p_go_max_inactivity=300
```

Figure 4.3: WiFi-Direct configuration file generated for a mobile station.

A significant limitation of our WiFi-Direct support is that active WiFi-Direct links are not handled appropriately in terms of signal propagation / signal strength modelling. The constraints for connection range are respected and handled without any issues, but the WiFi-Direct connections always work at a high bandwidth regardless of range. The issue is that we were not able to generalize the `wmediumd` interface for WiFi-Direct links. We found too intricate to do this, as it involves changes/patches at several spots in the Mininet-WiFi code. To mitigate this loss of fidelity, we calibrated the experiments described in Chapter 5 such that only short data transfers occurred, trying to minimize the observable difference between WiFi and WiFi-Direct links.

4.2.3 Cloudlet support

We model cloudlets as mobile stations that are fixed in space and in spatially close to an AP, by just making use of the base Mininet-WiFi API. This is done by our bootstrap script, illustrated in the code sketch of Figure 4.4, that shows that, if cloudlets are enabled, one cloudlet per AP will defined at the same position. By defining an explicit position for the “cloudlet station” (as shown) any mobility settings will not affect it, unlike for normal mobile stations, and the simulation engine will guarantee that the cloudlet will connect and stay connected to the AP.

This solution, while simple, does not exactly model the nature of a cloudlet. The more proper

approach we had in mind was to model model cloudlets as static hosts connected by Ethernet to the AP. In principle, the functionality should work as defined by the base Mininet framework (upon which Mininet-WiFi builds on). We found that did not work after consulting with the Mininet-WiFi developers and trying various different configurations.

The other limitation of our approach is that we are unable to enable cloudlets connected by a mesh network, as in the UGR scenario. This was not possible in the time scope of this thesis. As in the case of WiFi-Direct, we would have to extend/patch the base Mininet-WiFi functionality, given that mesh networking does not work out-of-the-box.

```
net = EdgeCloud( ... )

for i in range(0, numberOfAPs):
  pos = ...
  ap = net.addAccessPoint(..., position=pos, ...)
  aps.append(ap)
  ...
  if ENABLE_CLOUDLETS == True:
     c = net.addStation(... , position=pos, ...)
     cloudlets.append(c)
  ...
```

Figure 4.4: Code sketch for cloudlet setup.

4.3 File Synchronization Service

4.3.1 Functionality

The organization of the mobile file synchronization service is shown in Figure 4.5. The service, invoked per each mobile station or cloudlet at simulation start-up, allows network peers to share files opportunistically.

The main aspects of the service are as follows:

- We model a local file system to store each host's files. In reality, the physical file system is the same for all peers and hosted at a shared root-level directory. A separate folder per each host in the root-level directory contains the files that pertain to that host.
- A peer may act as content producers in automated manner. Local file generation within the service is governed in frequency by a pseudo-random Poisson process plus a constant value parameter for file size. The content of files is obtained by reading from the `/dev/urandom` device, thus files are merely random byte-streams.
- Files are synchronized between peers using the classic `rsync` POSIX utility, that is able to transfer only changed or new files between two hosts. The synchronization works by

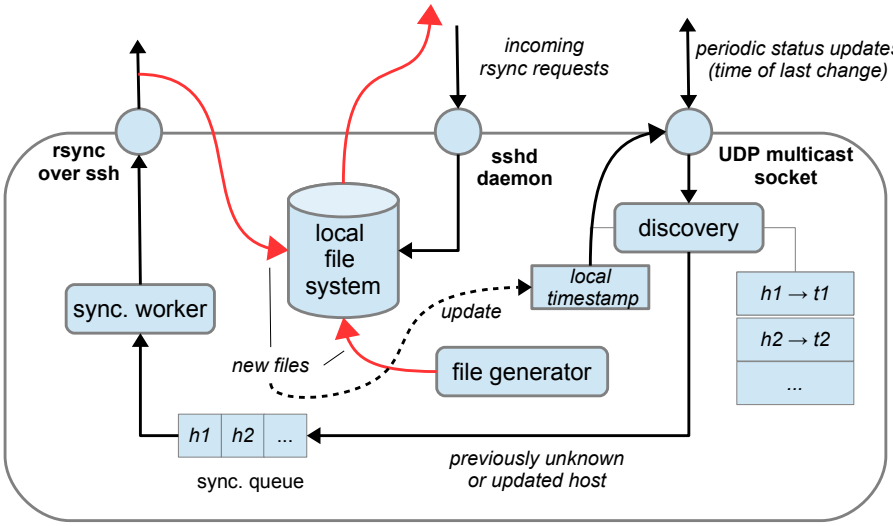


Figure 4.5: Overview of the mobile file synchronization service.

invoking `rsync` to fetch new content from another peer. This is done over a SSH connection, thus the service maintains a running `sshd` daemon.

- To make synchronization efficient and opportunistic, the service periodically reports the time-of-last-change in its local filesystem through multicast UDP. This allows for peers to discover other nearby peers dynamically, and know when changes took place and synchronization is necessary through the timestamp information.
- Requests for synchronization are queued and at most one active `rsync` process is active per peer. There might be multiple incoming `rsync` connections at a given time.

4.3.2 Implementation details

The service is implemented as a Python program with two concurrent threads, plus a `sshd` daemon process. The main thread is responsible for managing the global life-cycle, while a secondary thread perform the specific job of executing queued synchronization requests using `rsync`.

4.3.2.1 Main lifecycle thread

Some of the most important aspects for the main thread's code are illustrated in Figure 4.6. As shown in the constructor (`__init__`), state is maintained in terms of file management and discovery modules (`FileService` and `Discovery`), a pseudo-random Poisson process for governing file generation (a `PoissonProcess` instance), and an action queue containing tasks that need to be executed in the future with some deadline (an `ActionQueue` instance).

The `run` method in Figure 4.6 encodes the main thread's execution. It begins by initializing all modules and setting up initial actions, and then loops executing scheduled tasks in the

action queue. The tasks are actually implemented by methods in the same class. For example, `generateFileAction` corresponds to the action of generating a file. We see that the associated code performs the tasks, then reschedules itself for some (relative) time in the future (according to the pseudo-random Poisson process). Other actions, like handling status updates received via UDP multicast (`handleStatusUpdatesAction`), or refreshing the file synchronization queue (`updateSyncQueueAction`).

```
class Main():
  def __init__(self, ...):
    ...
    self.fs = FileService(sta, root_dir)
    self.disc = Discovery(sta, root_dir)
    self.rng = random.Random(int(SEED) + hash(self.sta))
    if (self.fileProducer)
      self.pp = poisson.PoissonProcess(rate=fileGenerationRate, rng=self.rng)
    self.aq = ActionQueue()
  def run(self):
    self.fs.start()
    self.disc.start()
    if (self.fileProducer)
      self.generateFileAction()
    self.handleStatusUpdatesAction()
    self.updateFileQueueAction()
    ...
    while self.aq.count() > 0:
      if not self.stop:
        self.aq.process()
  def generateFileAction(self):
    ...
    self.fs.generateFile(FILE_SIZE)
    self.disc.updateTimestamp()
    t = self.pp.timeTillNextEvent()
    self.aq.schedule(self.generateFileAction, int(t))
  def handleStatusUpdatesAction(self):
    self.disc.handleStatusUpdates()
    self.aq.schedule(self.sendAnnounceAction, 2)
  def updateSyncQueueAction(self):
    ...
    self.aq.schedule(self.updateSyncQueueAction, 5)
```

Figure 4.6: File synchronization service: code fragments for main thread.

4.3.2.2 The action queue

The core code of `ActionQueue` is shown in Figure 4.7. We see that the datatype implements an earliest-deadline first scheduler: the `schedule` method is used adds a deadline-task pair to a priority queue, while the `process` method delays execution until the deadline for the next task is

due and then executes that task.

```
import heapq
import time
import os

class ActionQueue(object):
   def __init__(self):
     self.heap = []
   ...
   def schedule(self, task, relativeDeadline=0):
     deadline = time.time() + relativeDeadline
     heapq.heappush(self.heap, (deadline, task))
   def process(self):
     if self.count() > 0:
       deadline,task = heapq.heappop(self.heap)
       delta = deadline - time.time()
     if delta > 0:
       time.sleep(delta)
     task()
```

Figure 4.7: The `ActionQueue` class.

4.3.2.3 File synchronization

The main code support for file synchronization between peers is shown in Figure 4.8. The `FileService` class has an associated queue for synchronization requests, and a worker thread, implemented in a class called `FileServiceWorker`, to handle these requests through invocations of `rsync`. The worker thread, as shown, loops by pulling requests from the `FileService` queue, and invoking `rsync` appropriately for each request. A `FileService` instance also manages a running `sshd` instance on port 2222 that is required for incoming synchronization requests.

```
class FileService:
  ...
  def __init__(self, sta, root_dir):
    ...
    self.syncQueue = Queue.Queue()
    self.worker = FileServiceWorker(syncQueue, ...)
    ...
  def start(self):
    ...
    worker.start()
    os.system("/usr/sbin/sshd -4 -p 2222")
  ...
  def generateFile(self):
    ...
    os.system("head -c %d /dev/urandom > %s; mv %s %s" \% (size, tmpFile,
        tmpFile, path))
    ...
  def addToSyncQueue(self, host, ip):
     self.syncQueue.put_nowait((host,ip))

class FileServiceWorker(threading.Thread):
  def __init__(self, queue, ...):
    self.queue = queue
    ...
  def run(self):
    while True:
      (host, ip) = self.queue.get(True)
      ...
      os.system("rsync -av %s --timeout=10 -e 'ssh -p 2222' root@%s %s" % ... )
  ...
```

Figure 4.8: File synchronisation support: code fragments.

Chapter 5

Evaluation

This chapter describes an evaluation of the simulation framework. We start by describing the simulation setup and parameters (Section 5.1. We then present the results from the multiple experiments we did, in which we compare the use of different features like the use of WiFi-Direct or cloudlets, as well as different connection handover policies and mobility pattern (Section 5.2),

5.1 Simulation setup

An initial configuration setup defines the main aspects of the simulation. With our configuration we are able to define the aspects such as number of nodes and their types, a choice of a network architecture based on the different variations that can be used (e.g., enabling/disabling cloudlets, WiFi-Direct groups, access points), we can manipulate the dimensions of the physical space and pattern that applies to the movement of mobile stations over time, manipulate the creation of file through distribution processes, parameters for tuning WiFi-Direct group formation (if is enabled) and positions of Access Points (APs).

The parameters are as follows:

1. Constant values common to all simulations:
    - **Duration** — Each simulation ran for 30 minutes.
    - **Area** — the spatial area has a dimension 500×500 meters.
    - **Mobile stations** — there were 50 stations, spread over the simulation area in random initial positions.
    - **File generation rate** — 1 file/minute per station, following a Poisson process law.
    - **File size** — fixed to 1 KB. We use a small value such that short data transfers occurred, trying to minimize the observable difference between WiFi and WiFi-Direct links, given our limitations in terms of WiFi-Direct data rate fidelity.

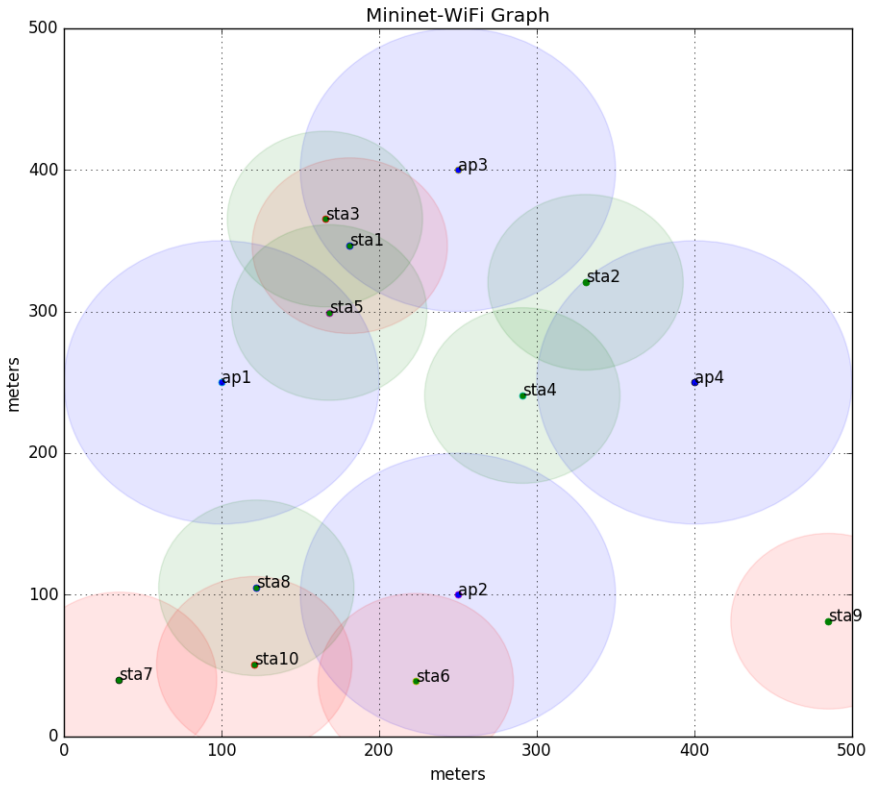


Figure 5.1: Spatial layout (Mininet-WiFi screenshot)

- **WiFi-Direct group management** — a station randomly becomes WiFi-Direct group owner every 30 seconds on average following a Poisson process, as long as it is not involved in a WiFi-Direct group already, and the group is dismantled after being empty for more than 30 seconds. The range is set to 60 meters.
- **Pseudo-random generator seed** — a constant seed for pseudo-random processes, such such that a number processes are repeatable for every simulation, including mobility, file generation and WiFi-Direct group management.

2. The parameters that could be enabled/disabled or changed depending on the experiment were:

   - **WiFi Access Points** — The optional use of 4 APs with fixed positions show in Figure 5.1 with a connection range of the 100 meters. Taking into account the area dimensions, this means that 50.26% of the simulation area was covered by WiFi.
   - **Cloudlets** — The optional use of 4 cloudlets, at the positions of APs.
   - **WiFi-Direct** — The optional use of WiFi-Direct.
   - **Mobility model** — two different mobility models could be considered, Random Waypoint and Random Walk, both configured with a maximum speed of 1 meter/second.

- **Handover policies** — Active and passive handover policies could be used. With active handover, a client station switches WiFi connections to an AP or WiFi-Direct group owner as soon as it finds another connection with closer range. In passive handover, a station sticks to a connection until it is lost.

5.2 Simulation results

In this section we present and analyse results for three sets of experiments. The experiments ran on an AMD FX X4 965 3.4GHz quad-core Linux machine with 8 GB of RAM using Ubuntu 16.04 as an operating system, with no computation running other than the simulation itself and the GNOME window manager, along with terminals necessary for triggering and monitoring the simulation.

In each experiment, we combined the simulation parameters to define 5 different types of network configuration, respectively making use of:

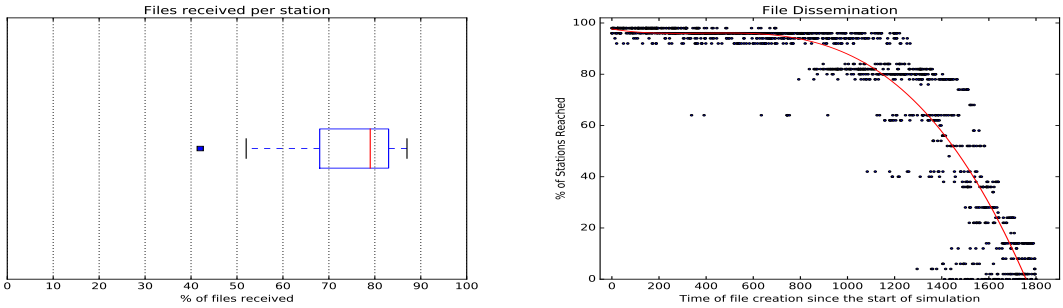
1. WiFi, WiFi-Direct, and cloudlets;
2. WiFi and WiFi-Direct;
3. WiFi and cloudlets;
4. WiFi only;
5. WiFi-Direct only.

In the first experiment set, the base setup, the active handover policy and a Random Waypoint mobility model are set. The second and third sets are variations of the base one: in the second set, the passive handover policy is set, and in the third a Random Walk mobility model is set. Overall, for all experiments, we examine the file dissemination rate attained. Then for the second and third experiment set, we respectively analyse the impact of handover policies with a connectivity analysis and of the change in mobility model.

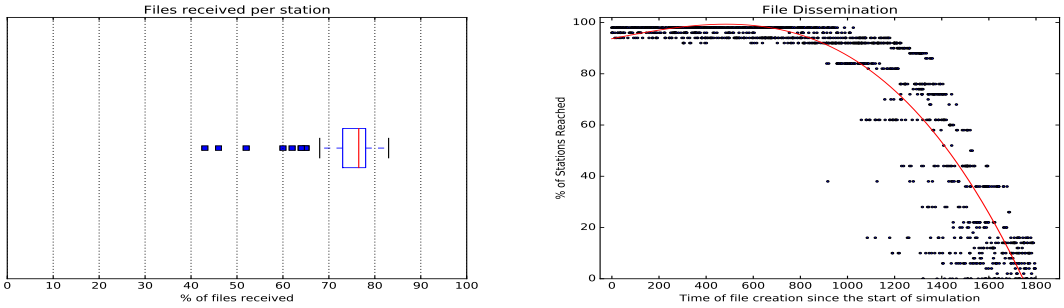
5.2.1 Base configuration results

In Figure 5.2, we depict results for file dissemination for each of the 5 network configurations. Per each configuration, the box plot refers to the percentage of all files generated held per station, which we call the dissemination rate, and the scatter plot depict the dissemination rates per each file in relation to its creation time along with a trend line.

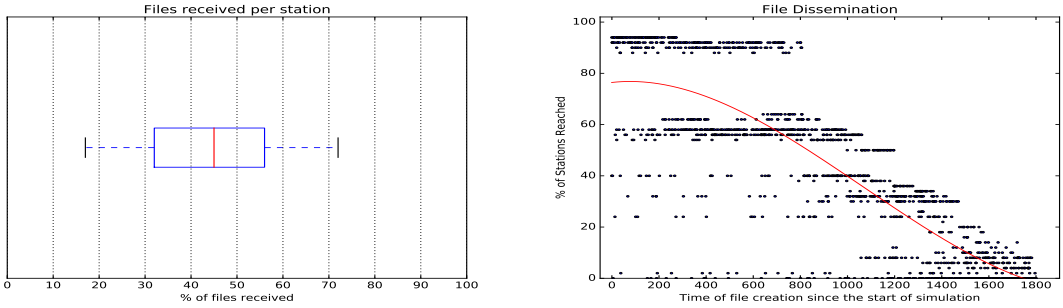
Looking at the results, we can draw some conclusions. First, it is possible to note the impact of WiFi-Direct, comparing the results of configurations (a) and (b) versus (c) and (d), where



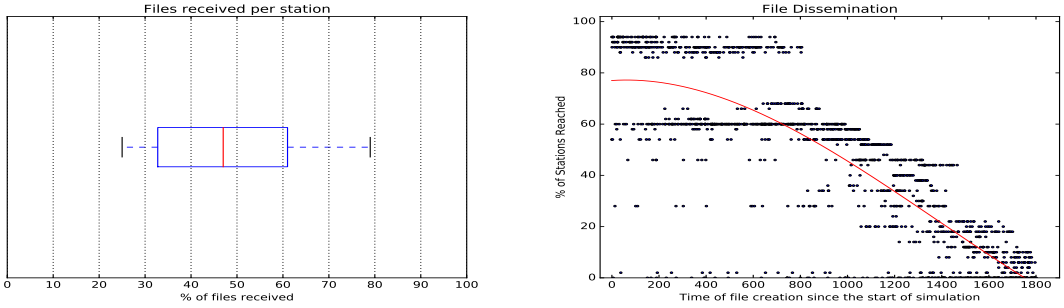
(a) WiFi-Direct + WiFi + Cloudlets



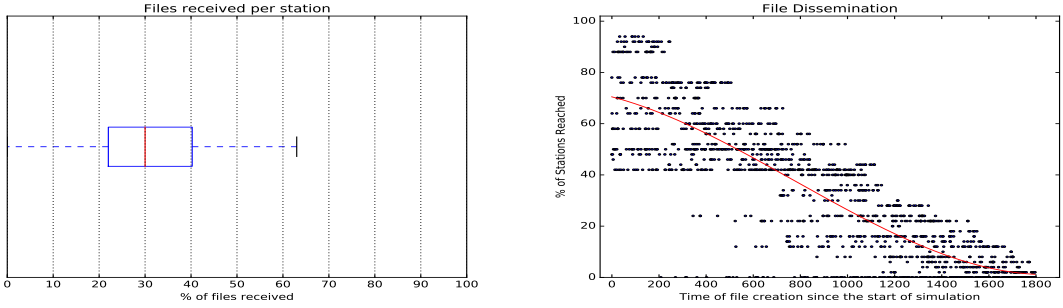
(b) WiFi-Direct + WiFi



(c) WiFi + Cloudlets



(d) WiFi

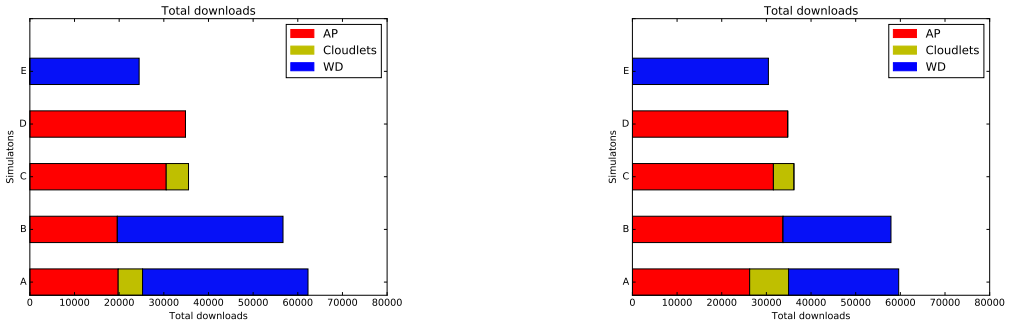


(e) WiFi-Direct

Figure 5.2: File dissemination – base configuration

we can see a difference of approximately 30% in dissemination rate, taking the median value as reference. With WiFi-Direct enabled, stations can exchange data among themselves outside of AP range, thus this significant difference is expectable. Comparing the use of only WiFi versus only WiFi-Direct, WiFi had a median rate of approximately 50% and WiFi-Direct of 30% only. Again this is expectable: the connection range is higher for APs than for WiFi-Direct groups, moreover APs provide a permanent service while WiFi-Direct groups can be transient. Regarding the use of cloudlets, comparing configurations (a) versus (b) and (c) versus (d), small improvements of approximately 2% in the median rate are observed. The small difference may be explained by a relatively small area for dissemination: stations will encounter each other frequently reducing the impact of cloudlet caching. Also, file synchronizations occurred in short bursts that could synchronize significant amounts of files, given the small file size we considered ($1KB$). Though we do not present concrete results, `rsync` invocations could transfer dozens of files in typically 1 or 2 seconds.

5.2.2 Passive handover



(a) Active handover (b) Passive handover

A: WiFi-Direct + WiFi + Cloudlets *B*: WiFi-Direct + WiFi *C*: WiFi + cloudlets *D*: WiFi *E*: WiFi-Direct

Figure 5.3: Total number of downloads

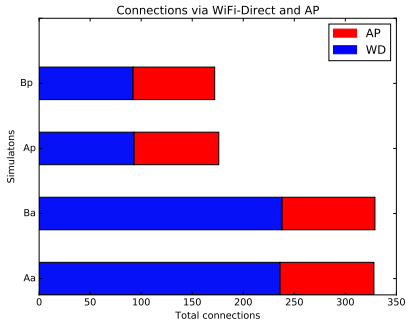
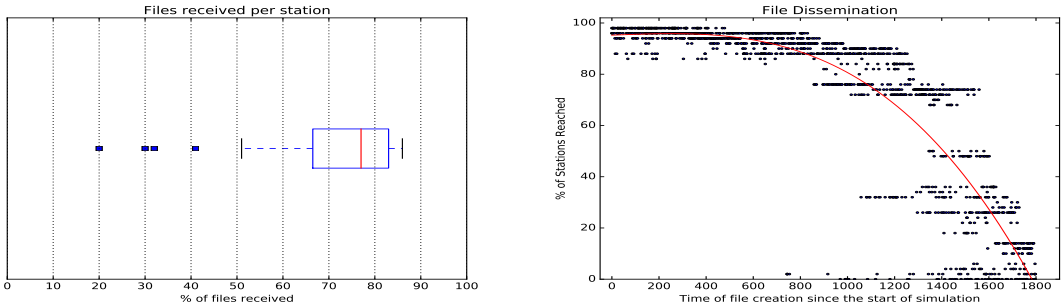
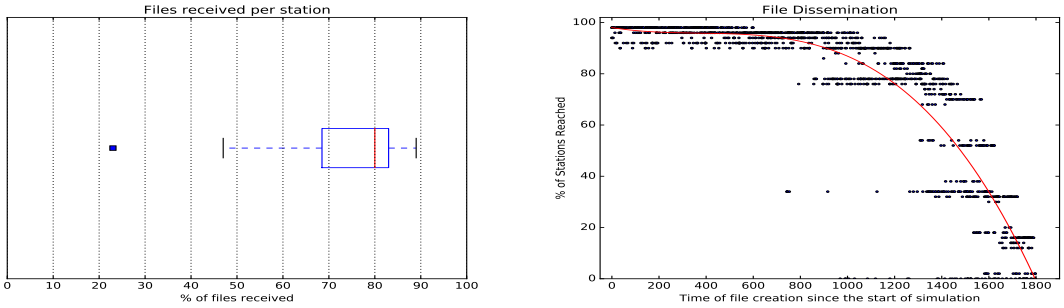


Figure 5.4: Connections to AP and WiFi-Direct

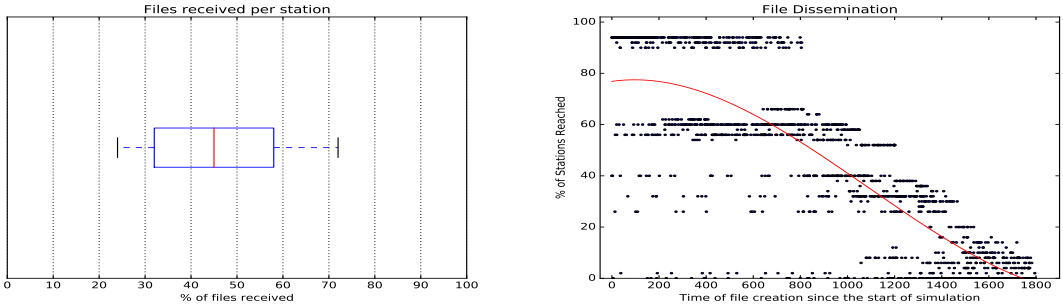
Aa: A + active handover *Ap*:A + passive handover *Ba*: B + active handover *Bp*: B + passive handover



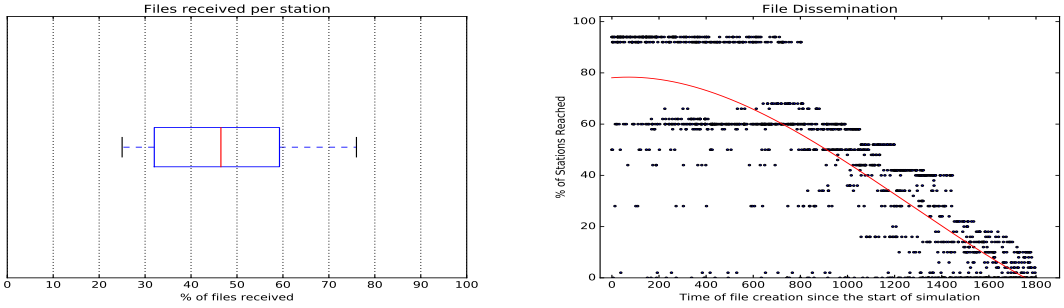
(a) WiFi-Direct + WiFi + Cloudlets



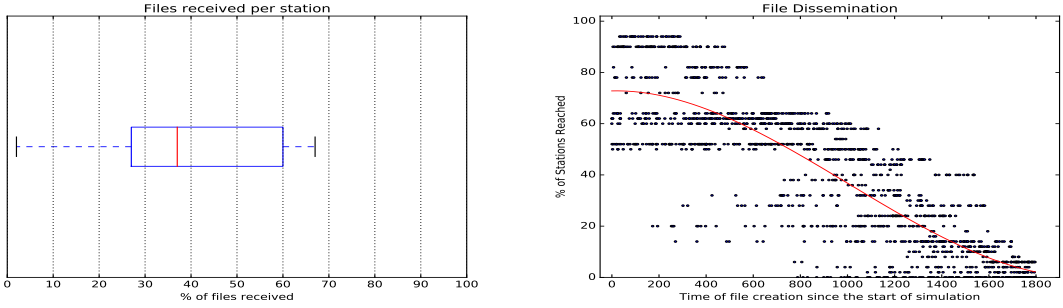
(b) WiFi-Direct + WiFi



(c) WiFi + Cloudlets



(d) WiFi



(e) WiFi-Direct

Figure 5.5: File dissemination – passive handover

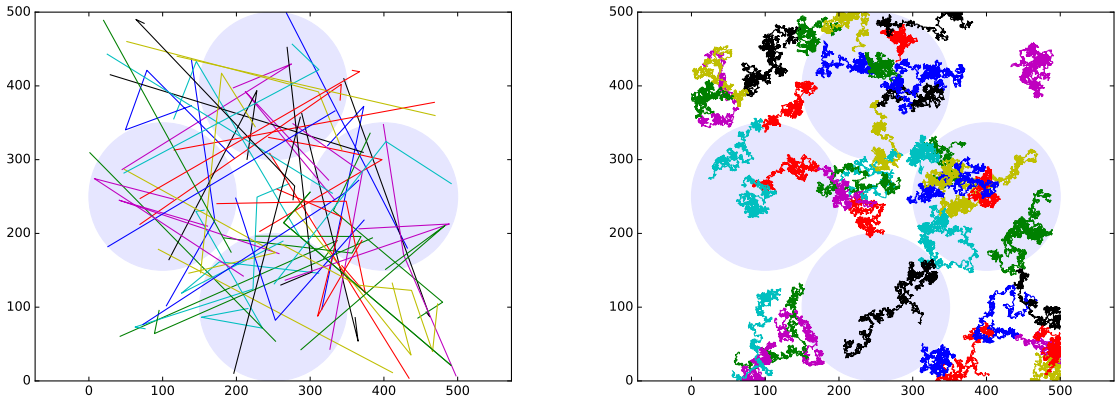
By changing the handover policy from active to passive, WiFi/WiFi-Direct connections are maintained while in connection range, and thus connection churn will be caused only by station mobility.

For both handover policies, we conducted an analysis to determine the number of connections and downloads via APs or WiFi-Direct GO, with results shown in Figure 5.3 and Figure 5.4. We intended to see how connections influence the dissemination of data, if more stations connected via WiFi-Direct would it return better or worse results. Given the results, we observed that it did not matter overall for the file dissemination rate. In the more aggressive handover there was a higher percentage of files transferred via WiFi-Direct, in the more passive there were more files transferred via AP but overall the same percentage files were downloaded. With this we can derive the conclusion that in order to get the best possible output of file dissemination, a connection must be had in order to synchronize files.

The file dissemination results are shown in Figure 5.5. Comparing to the base configuration results (Figure 5.2, discussed previously), there are small changes to observe, with a slight increase or decrease in dissemination rate in almost all configurations except for the WiFi-Direct only (e) configurations, where passive handover visibly yields better results.

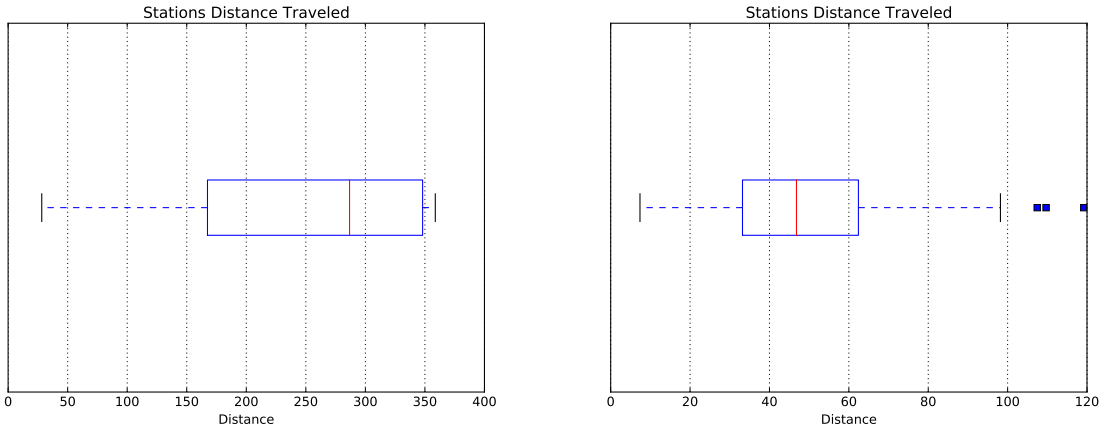
5.2.3 Alternative mobility pattern

By changing the mobility model from Random Waypoint to Random Walk, the movement of stations will likely have much lower amplitude. Figure 5.6 shows the movement of the stations using both patterns. It is possible to observe that, with Random Walk mobility model, stations roam a much smaller area. In Figure 5.7 we show a box-plot for the distance between the initial and final position of the stations, that also confirm this: approximately 50 meters for Random Walk versus 300 meters for Random Waypoint. The file dissemination results for Random Walk are in turn shown in Figure 5.8, where it is possible to see the results are a lot worse than those of all the others simulations presented. This results show that, as expected, the magnitude of mobility of a station increases the file dissemination rate.



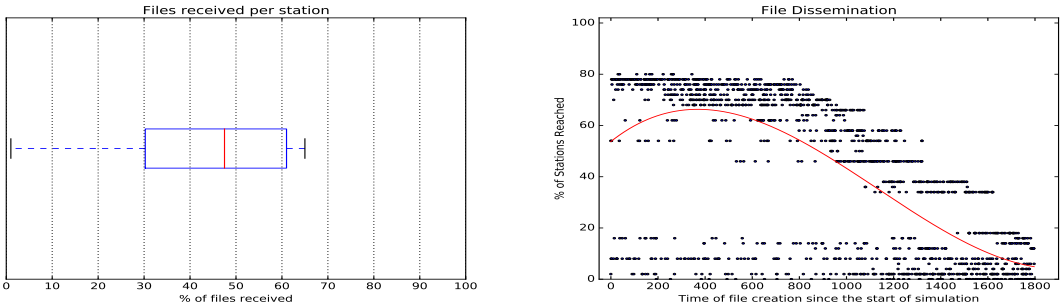
(a) Random Waypoint (b) Random Walk

Figure 5.6: Station positions over time

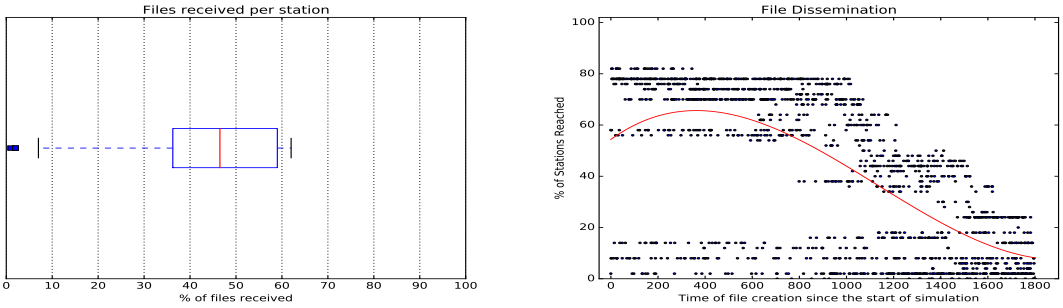


(a) Random Waypoint (b) Random Walk

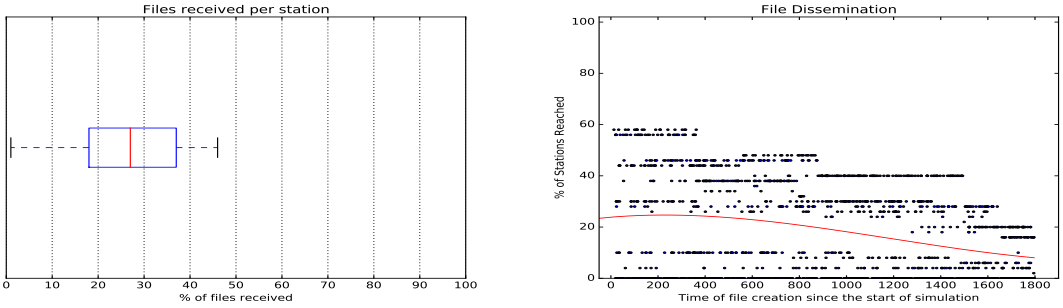
Figure 5.7: Distance between initial and final positions.



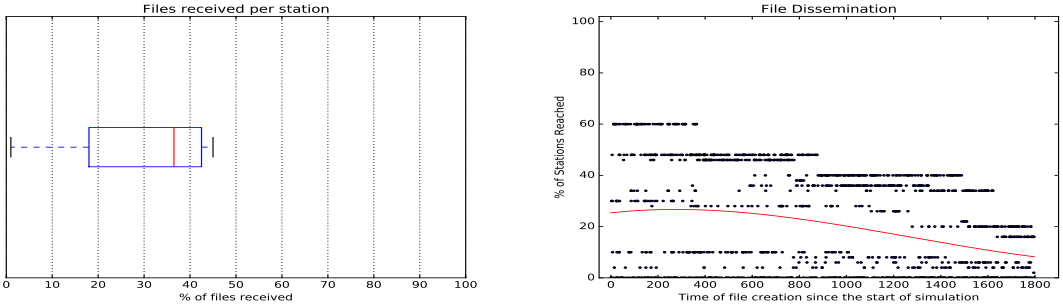
(a) WiFi-Direct + WiFi + Cloudlets



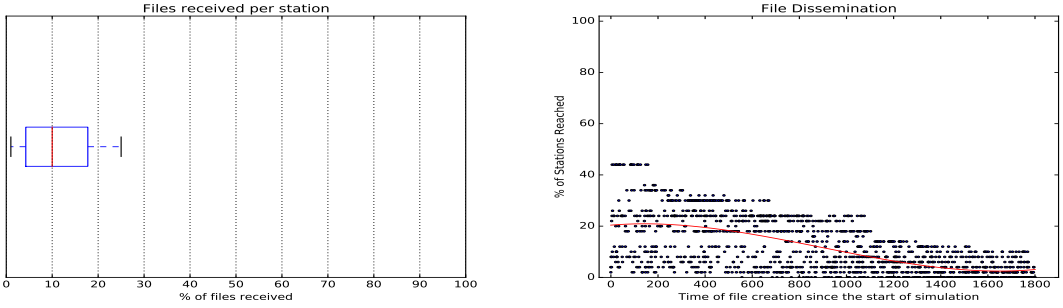
(b) WiFi-Direct + WiFi



(c) WiFi + Cloudlets



(d) WiFi



(e) WiFi-Direct

Figure 5.8: File dissemination – Random Walk mobility model

Chapter 6

Conclusions

6.1 Discussion

The ubiquity of mobile devices led to the consideration of mobile edge-clouds, networks formed by devices in close proximity without infrastructural support, or at most optionally complemented by cloudlets. In this thesis we addressed the problem of simulating applications at scale by extending the Mininet-WiFi framework.

The Mininet-WiFi extensions comprise WiFi-Direct group formation and the modelling of cloudlets. In addition, we developed a mobile synchronization service that runs within the Mininet-Wifi environment to model some of the core features of the User-Generated Replays (UGR) Hyrax case-study. These developments were evaluated with different simulations accounting for different parameterisations for network setup, user mobility, and connection handover. In particular, the evaluation covers the use of WiFi or Wifi-Direct alone versus the use of WiFi with WiFi-Direct, and with/without the use of cloudlets.

6.2 Future Work

This work presents a number of interesting directions to explore in future work:

- A number of limitations in our framework can be addressed. In particular, improving simulation fidelity for the bandwidth speed of WiFi-Direct data transfers, and the support of mesh networking would allow us to model every aspect of the network in the UGR case-study.
- A more extensive evaluation of our base simulation scenario can be made. It may consider different parameterisations for the number of mobile stations and access points, the dimension of the simulation area and WiFi AP positions in that area, the mobility pattern, or the file generation rate.

- Our base file synchronization model can be refined, by considering probabilistic processes for aspects such as file consumption or for the popularity of content. This can be coupled as well with alternative strategies for WiFi-Direct group formation and connection handover.
- The consideration of other case-study applications, for instance ones that involve content sharing in different scenarios or employ different content sharing models (e.g., communications in disaster scenarios), or applications that involve computation as well as communication.

Bibliography

[1] The hyrax project. Online. Acessed December 2017.

[2] mac802.11_hwsim website. Online. Acessed December 2017.

[3] Tracy Camp, Jeff Boleng, and Vanessa Davies. A survey of mobility models for ad hoc network research. *Wireless communications and mobile computing*, 2(5):483–502, 2002.

[4] Min-Cheng Chan, Chien Chen, Jun-Xian Huang, Ted Kuo, Li-Hsing Yen, and Chien-Chao Tseng. Opennet: A simulator for software-defined wireless local area network. In *Wireless Communications and Networking Conference (WCNC), 2014 IEEE*, pages 3332–3336. IEEE, 2014.

[5] Ramon dos Reis Fontes and Christian Esteve Rothenberg. Mininet-wifi: A platform for hybrid physical-virtual software-defined wireless networking research. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 607–608. ACM, 2016.

[6] Ramon Dos Reis Fontes, Mohamed Mahfoudi, Walid Dabbous, Thierry Turletti, and Christian Rothenberg. How far can we go? towards realistic software-defined wireless networking experiments. *The Computer Journal*, pages 1–14, 2017.

[7] Ramon R Fontes, Samira Afzal, Samuel HB Brito, Mateus AS Santos, and Christian Esteve Rothenberg. Mininet-wifi: Emulating software-defined wireless networks. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 384–389. IEEE, 2015.

[8] Antoine Fraboulet, Guillaume Chelius, and Eric Fleury. Worldsens: development and prototyping tools for application specific wireless sensors networks. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, pages 176–185. IEEE, 2007.

[9] Harald T Friis. A note on a simple transmission formula. *Proceedings of the IRE*, 34(5): 254–256, 1946.

[10] Thomas R Henderson, Sumit Roy, Sally Floyd, and George F Riley. ns-3 project goals. In *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, page 13. ACM, 2006.

[11] Xiaoyan Hong, Mario Gerla, Guangyu Pei, and Ching-Chuan Chiang. A group mobility model for ad hoc wireless networks. In *Proceedings of the 2nd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 53–60. ACM, 1999.

[12] W-J Hsu, Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Ahmed Helmy. Modeling time-variant user mobility in wireless mobile networks. In *INFOCOM 2007. 26th IEEE international conference on computer communications. IEEE*, pages 758–766. IEEE, 2007.

[13] M. Imran, A. M. Said, and H. Hasbullah. A survey of simulators, emulators and testbeds for wireless sensor networks. 2:897–902, June 2010. ISSN: 2155-8973. doi:10.1109/ITSIM.2010.5561571.

[14] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.

[15] Katerina Pechlivanidou, Kostas Katsalis, Ioannis Igoumenos, Dimitrios Katsaros, Thanasis Korakis, and Leandros Tassiulas. Nitos testbed: A cloud based wireless experimentation facility. In *Teletraffic Congress (ITC), 2014 26th International*, pages 1–6. IEEE, 2014.

[16] Dipankar Raychaudhuri, Ivan Seskar, Max Ott, Sachin Ganu, Kishore Ramachandran, Haris Kremo, Robert Siracusa, Hang Liu, and Manpreet Singh. Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 3, pages 1664–1669. IEEE, 2005.

[17] J. Rodrigues, E. R. B. Marques, J. Silva, L. Lopes, and F. Silva. Video dissemination in untethered edge-clouds: a case study. In *Proc. 18th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS)*, DAIS'18, pages 137–152. Springer, 2018.

[18] João Rodrigues, Eduardo RB Marques, Luís MB Lopes, and Fernando Silva. Towards a middleware for mobile edge-cloud applications. 2017.

[19] J. A. Silva, H. Paulino, J. M. Lourenço, J. Leitão, and N. Preguiça. I Know What You Did Last Summer: Time-Aware Publish/Subscribe for Networks of Mobile Devices. *CoRR*, abs/1801.00297, 2017.

[20] Joaquim Silva. Simulation of algorithms for mobile ad-hoc networks. In *Master Thesis*. Faculdade de Ciências do Porto, 2015.

[21] Joaquim Silva, Daniel Silva, Eduardo RB Marques, Luís Lopes, and Fernando Silva. P3-mobile: Parallel computing for mobile edge-clouds. In *Proceedings of the 4th Workshop on CrossCloud Infrastructures & Platforms*, page 5. ACM, 2017.

[22] Pedro M Pinto Silva, Joao Rodrigues, Joaquim Silva, Rolando Martins, Luís Lopes, and Fernando Silva. Using edge-clouds to reduce load on traditional wifi infrastructures and improve quality of experience. In *Fog and Edge Computing (ICFEC), 2017 IEEE 1st International Conference on*, pages 61–67. IEEE, 2017.

[23] Statista. Number of smartphone users worldwide from 2014 to 2020 (in billions). Online, June 2016. Acessed December 2017.

[24] Mininet Team. Mininet: An instant virtual network on your laptop (or other pc), 2012.

[25] Onelab Future Internet Testbeds. R2lab testbed. https://r2lab.inria.fr/index.md, 2016.

[26] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[27] Alexander Zimmermann, Mesut Günes, Martin Wenig, Jan Ritzerfeld, and Ulrich Meis. Architecture of the hybrid mcg-mesh testbed. In *Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization*, pages 88–89. ACM, 2006.