Simulation of Algorithms for Mobile Ad-Hoc Networks

Joaquim Magalhães Esteves da Silva
Mestrado em Ciência de Computadores
Departamento de Ciência de Computadores
2015

Orientador
Fernando Manuel Augusto da Silva, Professor Catedrático,
Faculdade de Ciências da Universidade do Porto

Coorientador
Luís Miguel Barros Lopes, Professor Associado,
Faculdade de Ciências da Universidade do Porto

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

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

O Presidente do Júri,

Porto, ______/______/_________

MSC

Agradecimentos

Em primeiro lugar gostaria de agradecer aos meus orientadores, Professor Doutor Fernando Silva e Professor Doutor Luis Lopes pela oportunidade única de trabalhar num projecto da envergadura do Hyrax, permitindo-me no ambito deste projecto passar um mês na Carnegie Mellon University em Pittsburgh, onde trabalhei com membros locais da equipa aos quais devo também agradecer, nomeadamente à Professora Doutora Priya Narashimhan, ao Utsav Drolia e ao Rolando Martins. Agradeço também aos restantes membros da equipa do Hyrax na Faculdade de Ciências da Universidade do Porto, que trabalharam neste projecto comigo.

Tenho também de agradecer à minha familia por todo o apoio que me deram ao longo destes últimos anos, apoio sem o qual nunca teria chegado onde cheguei.

Finalmente, um agradecimento especial à minha amiga Ana Sofia, por me acompanhar durante todo o periodo de desenvolvimento desta dissertação, dando-me apoio moral, técnico e ajudando-me a escrever este documento.

Abstract

The popularity of mobile gadgets, whether *Smartphones*, *Tablets* or others, has increased to a point in which their number is said to have surpassed the number of people. This has made the formation of mobile ad-hoc networks an appealing and popular functionality, particularly in situations in which Internet or even Wi-Fi access may have been restrained or fully closed. The Hyrax project is a good example of an initiative to promote similar functionalities to users as it aims to build computing edge clouds from the crowd-sourcing of mobile devices. This is of special interest in large gatherings of people such is the case in sports venues. The density of devices stresses available infrastructures, thus impacting all users in their use of Wi-Fi connections to access Internet, social networks, or the feeds of game replays. To envisage such goals, one needs to fully understand the potential and limitations of available device-to-device communications technologies.

The main goal of this dissertation is to study the advantages and the limitations of current wireless communication technologies, namely Wi-Fi, Wi-Fi Direct, TDLS, and Bluetooth. This is of special interest when the number of devices increases significantly. To accomplish our goal, a set of use cases and scenarios were prepared, real experiments were performed to gather data that could allow us to instrument a simulator to take the study further. We instrumented a NS-3 simulator in order to create realistic experiments on the scalability of the different wireless technologies that could allow us to draw conclusions. Experiments were prepared, both to involve the use of distinct network overlays to organize mobile devices and distinct replication strategies. We also study the effect of exchanging Wi-Fi connections to a potentially saturated access point by having local communications via Wi-Fi Direct or TDLS.

Resumo

A popularidade de *gadgets* móveis, sejam eles *Smartphones*, *Tablets* ou outros, aumentou a ponto de o seu número ultrapassar o número de pessoas. Isto fez com que a formação de redes *ad-hoc* se tenha tornado numa funcionalidade apelativa e popular, em particular em situações onde o acesso à Internet ou ao *Wi-Fi* é restringido ou totalmente fechado. O projeto Hyrax é um bom exemplo de uma iniciativa para promover tais funcionalidades para os utilizadores, já que o seu objetivo é desenvolver uma plataforma *edge-cloud* construída a partir de *crowd-source* de dispositivos móveis. Isto tem especial interesse em grandes aglomerações de pessoas, tal como acontece em recintos desportivos. A densidade de dispositivos sobrecarrega as infraestruturas disponíveis, tendo desta forma um impacto nos seus utilizadores ao utilizar conexões *Wi-Fi* para aceder à Internet, redes sociais ou fontes de repetições dos jogos. Para atingir tais objetivos, é necessário compreender na totalidade o potencial e as limitações das tecnologias de comunicação.

O principal objetivo desta dissertação é estudar as vantagens e as limitações das tecnologias para comunicação sem fios atuais, nomeadamente *Wi-Fi*, *Wi-Fi Direct*, *TDLS* e *Bluetooth*. Isto é de especial interesse quando o número de dispositivos cresce significativamente. Para atingir o nosso objetivo, um conjunto de casos e cenários foram preparados, e experiências reais foram realizadas para obter dados que nos permitiram instrumentar o simulador para continuar o estudo. Programamos o simulador NS-3 de forma a permitir-nos criar experiências realísticas para estudar a escalabilidade de diferentes tecnologias sem fios que nos permitiram tirar conclusões. Foram também preparadas experiências para envolver várias estratégias de *overlay* para organizar dispositivos e também estratégias de replicação. Estudamos também o efeito de trocar comunicações *Wi-Fi* num potencialmente saturado ponto de acesso, tendo comunicações locais com *Wi-Fi Direct* e *TDLS*.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For a long time, people have been trying to improve their capacity to solve problems. From the Chinese that used an abacus to make their calculations easier, all the way to the modern era, where the computer reigns as a tool to allow us to compute ever more complex problems.

The computer itself has suffered a gigantic evolution over the last century, and with it, many new technologies have emerged, with recent examples being the Internet, and wireless and sensor technologies. These have also evolved very rapidly, not just in speed, but also in capacity and connectivity. We reached a point in which these technologies are part of our everyday life.

Technology development has increased our capacity to solve larger and more computational demanding problems. That has been accomplished with parallel and distributed powerful computing machine clusters that result from the aggregation of many smaller computing entities (e.g. CPUs, GPUs), which are seen as a unified computing platform. Grid Computing and more recently Cloud Computing are evolutions of such paradigm in which emphasis is increasingly put on abstracting the infrastructure, both in CPU and storage, its use and management, through virtualization and new software as a service paradigm that expand and generalize the use of computing [14].

While these new offerings are very appropriate for businesses, with the developments in mobile technology with devices computationally more capable, with larger storage capacity, and more efficient in wireless communication, cloud computing offerings are being used as a backbone for mobile applications. The innovations in wireless technology have led to a new world of opportunities with what is now described the Internet of Things (IOT).

The number of mobile devices such as smartphones and tablets has increased dramatically, their numbers surpassing the 3.5 billion [22]. Their capacity both, in terms of computing power and storage have also improved to a point in which current devices are more powerful than personal computers of the early 2000s. The number of these devices, and their capacity, is expected to continue growing worldwide, and thus they have become a desirable platform to deploy Cloud systems, allowing for new applications to emerge.

This dissertation is developed within the context of the Hyrax Project [3], whose goal is to create computing edge-clouds that run on mobile devices, thus taking full advantage of the resources found in new smartphones and tablets, and allowing developers to create new and innovative applications. By building edge clouds, the user experience with many applications, specially those functioning in high density of devices and resorting to a stressed physical infrastructure, could benefit by offloading some of the computation from their servers to the mobile cloud, where users can crowd-source resources. This requires a combination of skills and technologies, namely in understanding on how wireless technologies scale *per se* and how their combination could improve scalability.

Understandably, studying the scalability of wireless communication technologies, such as Wi-Fi, Wi-Fi Direct, Bluetooth, and TDLS, require the use of simulation and that will be the focus of this work.

1.1 Cloud Computing

Cloud Computing is a very popular term nowadays and can have a broad meaning. The Cloud is a metaphor for Internet, and the term "Cloud Computing" can be seen as storing and accessing data and programs that run on the Internet instead of the local computer. These services typically run on large data centers offering its users on-demand computing resources and a large variety of services [14].

There are many types of clouds offering different services to the costumers. Clouds services can be categorized as *Infrastructure-as-a-Service (IaaS)*, *Platform-as-a-Service (PaaS)* or *Software-as-a-Service (SaaS)*. These services can also be public, private, community or hybrid [34].

Platform-as-a-Service cloud systems provide the consumer with a platform where they can deploy consumer-created or acquired applications. On PaaS clouds, users do not have control over the underlying system (network, servers, operating systems, etc...), but can control deployed applications and possibly change configuration settings for the application-hosting environment.

When users need a system for processing, storage, networks and other fundamental resources to deploy and run arbitrary software, the Infrastructure-as-a-Service cloud. In IaaS the user does not control the underlying infrastructure, but has control over operating systems, storage, deployed applications and may have limited control over selected networking components.

The Software-as-a-Service provides consumers with applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as web browser or a program. In this kind of cloud, users can only control, in some cases, user-specific application configuration settings.

As for the types of cloud deployment models, private clouds are provisioned for exclusive use by a single organization comprising multiple consumers and can be owned, managed, and operated by the organization, a third party, or some combination of them. Community cloud infrastructure are for exclusive use of a community of consumers from organizations that have shared concerns and as with private clouds can be owned, managed, and operated by one or more of the participating organizations a third party or a combination of them.

Public clouds provide services to the general public and may be owned, managed, and operated by a business, academic, or government organization, or some combination of them.

Finally, hybrid clouds are composed of two or more distinct infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability.

1.1.1 Edge-Clouds

The term *Edge Cloud* comes from the junction of *Edge Computing* and *Cloud Computing.* Edge Computing is pushing the frontier of computing applications, data and services away from centralized nodes to the logical extremes of a network. This approach requires leveraging resources that may not be continuously connected, such as smartphones or tablets.

This *Edge* concept can be applied to a wide range of technologies such as wireless sensor networks (WSN), distributed peer-to-peer ad-hoc networking, Grid/MESH computing or even Cloud computing.

With the fast increase of smartphone usage, applying *Edge Computing* to *Cloud Computing* is a natural step to be taken from the traditional infrastructures.

But porting systems to a decentralized network where the node availability is inconstant (*churn* effect), the communication capabilities are limited and energy consumption is a problem, posing new challenges that have to be faced and overcome. This is one of the challenges of the Hyrax Project that aims to develop an edge-cloud capable of running on user owned smartphones.

1.2 Problem Definition and Proposed Approach

Developing edge-clouds with mobile devices brings plenty of additional challenges compared to the typical Cloud setting. Problems resulting from the mobility of nodes, entering and leaving the network (*churn*), battery usage (or energy efficiency), limited hardware and software resources, and technologies available, become of major importance and need to be addressed.

In the context of the Hyrax project, we created several deployment scenarios, such as firefighter ad-hoc communication, museum guides or an on-demand content provider network for sport venues. This last scenario is the one we chose to start the development of this project as it is possibly the one posing most technical challenges.

1.2.1 The Problem

Everyone that attends, or has ever attended, a sports game has certainly experienced the wish for immediate access to a replay of an important play that he might have missed, or simply because just wants to review it. Having this kind of facility naturally boosts a supporter experience during the game. To address this problem, sports teams have been implementing in their stadiums Wi-Fi access to provide Wi-Fi access to their supporters, and have also teamed up with software companies, such as Yinzcam [8] for providing live content replays. The current approach relies only on infrastructure network that requires a large number of Access Points to reach a high number of supporters, thus making these installations very expensive while not providing sufficient bandwidth for all the supporters.

Techniques such as Multicast [15] have been implemented to increase the quality of service provided and the number of simultaneous connections. While these techniques provide solutions for official content generation, such as live video streaming, they do not solve all problems, particularly those that arise from the limitations of overcrowded access points. Furthermore, they do not easily accommodate the sharing of user generated photos or video content.

The Hyrax project addresses this problem by proposing the use of ad-hoc technologies to crowd-source an edge-cloud that is capable of increasing the number of supporters with Internet access, providing caching for on-demand content and even streaming live content from any section of the venue. To accomplish this it needs to resort to different wireless technologies such as Wi-Fi, Wi-Fi Direct, Bluetooth and TDLS, different network organization schemes and dissemination algorithms. Understanding the limitations and scalability of the wireless technologies and how their combined use may affect bandwidth, performance and battery usage is crucial to our research. Naturally, one must use simulation no only due to the limitations on the number of smartphone and tablet made available to us, but also the difficulty in reproducing specific scenarios and implementing all approaches on real devices.

1.2.2 Goals and Contributions

The main goals of this work are centered on the evaluation of the limitations of existing wireless communication technologies in supporting content distribution in ad-hoc networks. We define a set of scenarios in which the mobile devices can communicate with content servers, fixed or mobile, using technologies such as Wi-Fi, Wi-Fi Direct, TDLS and Bluetooth, to download content. We evaluate the performance of the wireless technologies in each of these scenarios through simulation, using download speed and scalability. To improve the accuracy of the experiments, the simulator was instrumented with parameter values gathered from a testbed composed of several physical devices (Android smartphones and tablets). Since content dissemination is a key issue in our experiments, we also vary the number of available content servers to reduce contention. Finally, we also experiment with distinct network overlays to assess which are adequate for use for content distribution in ad-hoc networks. The aforementioned scenarios were implemented in the simulator at the application level in the OSI Stack. This means that, for example, we are implementing overlay networks at the user level. We do this because our goal in this content distribution scenario is to use non-rooted devices.

The work addressed in this dissertation makes the following contributions:

- A detailed description of the existing wireless technologies that enable the creation of ad-hoc networks, and a comparative study of the state of the art simulators for wireless networks;
- Instrumentation of the NS-3 simulator to simulate Bluetooth, Wi-Fi Direct and TDLS based on values obtained with testbed of physical devices;
- Definition of realistic scenarios and their evaluation using each of the wireless technologies individually and in combination, both in cases were content is coming form a fixed server or from mobile servers;
- Simulation of different network overlays, e.g., ring, star, P2P, and scatternet, to evaluate scalability using metrics such as average time for file discovery, average number of downloads per time unit, locality and churn resilience.

1.3 Dissertation Outline

This document is organized in six chapters, starting from with this introduction where the motivation and goals of this work were explained, and contributions highlighted.

The first two chapters introduce relevant concepts and detailed research on the state of the art on the networks, ad-hoc communications and simulation.

Chapter 2 describes the concept of ad-hoc and mesh networks and their routing protocols, and an overview on the state of the art of the projects developed with these networks is done. Within this chapter there is also a small introduction to the OSI model in order to better explain in which layer most of the work was done, and why it is important to understand all the layers in this project.

Chapter 3 is on the subject of simulation, including simulator tools and simulation techniques. This chapter starts with an introduction to simulation as a way of testing different theories and systems, followed by a description the different types of simulations, and of course, various simulation tools were described and analyzed. By the end of this chapter, a comparison between the most promising tools is made, and the simulator used for the later chapters is chosen.

The chapter 4 talks about the second part of our work, where the objective was to validate the results obtained with the simulator used by comparing the simulated results to experiments using real devices with multiple scenarios and technologies.

Chapter 5 is a study of different overlays using current wireless technologies for the problem defined before. The purpose of this chapter is to compare different approaches for reducing the AP/Server load in terms of scalability, delays and *churn* resilience.

Finally chapter 6, we present our conclusions on the work developed and discuss some future work that can be done and matters for the future of the Hyrax project.

Chapter 2

Networks and Protocols

Since the beginning of the Internet, computer scientists have been developing new and innovative ways to facilitate the communication between devices by building many different types of computer networks and using different approaches to tackle related issues such as routing, overlay organization, etc.

Initial computer networks were static in their configuration and mostly built upon existing infrastructures and used cabled connections, using either modems, routers or point-to-point connections. These networks were limited in terms of mobility and thus restricted to a very stable topology.

More recently, with the fast development of wireless technologies, networks have become dynamic and more powerful, with new types of device-to-device communication emerging. In a traditional setup, multiple devices connect to the Internet, and can talk with each other using an access point that takes care of the organization and forwarding of messages in the network. To overcome limitations on these networks new device-to-device communication technologies emerged such as ad-hoc or Wi-Fi Direct have appeared, thus allowing for novel applications.

This chapter addresses these wireless networks (mobile), relevant routing protocols, and also describes projects currently available that make use of these technologies.

2.1 TCP/IP, OSI and Bluetooth Protocol Stack

The process of exchanging data between network devices can be a complex process for many reasons. To simplify and create standards on how data should be sent across devices, different models have been developed, grouping the different steps data has to go through to be sent or received in different layers that form a stack. In these models, when data is sent, each data packet is formed and modified from the top layer, all the way to the bottom one, where it is ready to be transfered. Upon reaching the destination device, the packet has then to go through all the layers, this time from the bottom, till it finally ends its transfer.

The *Open Systems Interconnection (OSI)* model is the most generic and used data communication model. It includes seven layers that can be mapped into two major categories: Application and Communication.

In each of the seven layers, this model defines a set of rules that must be followed by manufacturers when developing hardware and software related with networks operations or services. In this model, the first 3 layers are related with communications between applications, while the other 4 are more related with data communication between two points in the network.

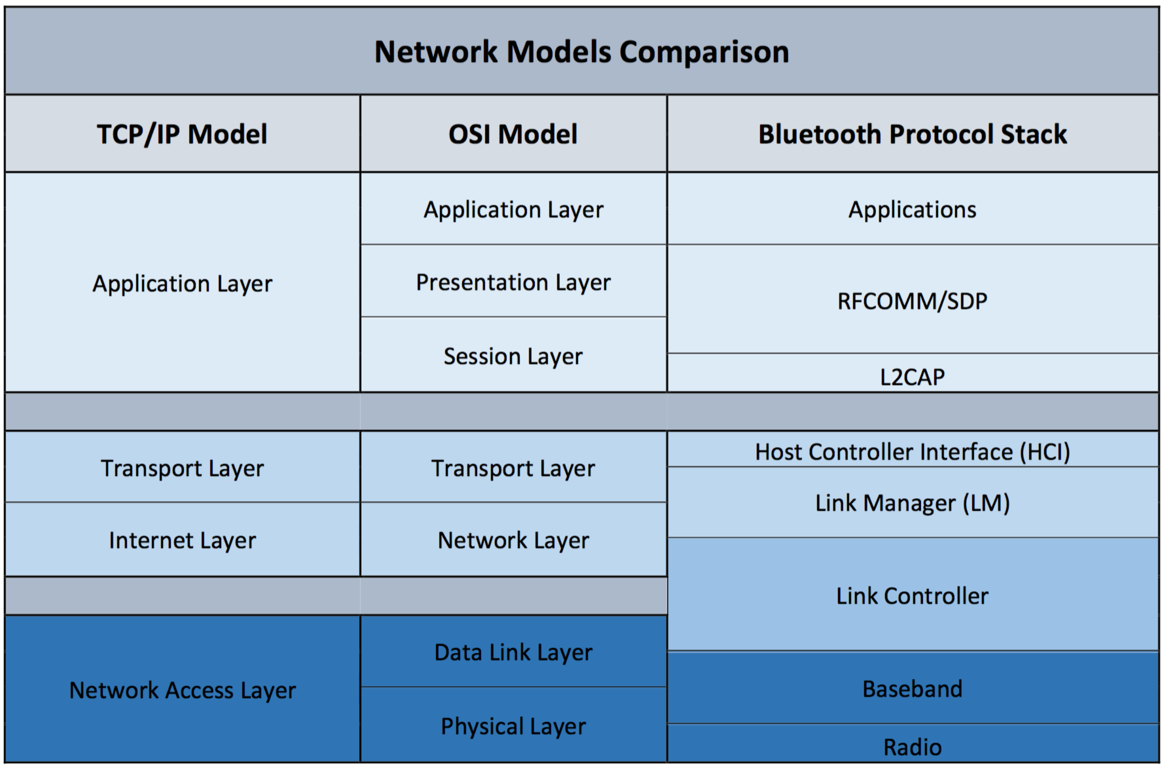
In this model there are two main characteristics that define the interconnection between layers:

- Each layer offers services to the upper layers
- Each layer can only communicate with homologous layers in other devices

Two adjacent layers must know the interfaces created between them. The interfaces specify that the layer $N + 1$ must send to the layer N and vice-versa, which means that in practice when a layer N receives information from a layer $N + 1$, it adds information according to the service it performs. This process of adding information while information travels down the OSI model layers, is know as *encapsulation*.

Using the same reasoning, when data reaches its destination device and travels up in the OSI model layer stack, the information added on each homologous layer is removed and when data reaches the application it will be the same as the original data sent.

This behavior defined by the OSI model is also used in other protocol stacks or models.


Network Models Comparison

TCP/IP Model	OSI Model	Bluetooth Protocol Stack
Application Layer	Application Layer	Applications
	Presentation Layer	RFCOMM/SDP
	Session Layer	L2CAP
Transport Layer	Transport Layer	Host Controller Interface (HCI)
		Link Manager (LM)
Internet Layer	Network Layer	Link Controller
Network Access Layer	Data Link Layer	Baseband
	Physical Layer	Radio

Figure 2.1: TCP/IP Stack compared with OSI stack

While OSI describes the common communication stack, for *Internet Protocol (IP)* and *Transmission Control Protocol (TCP)* a simplified version was made containing only 4 layers, abstracting in this way some of the layers thus facilitating the classification and organization of network services.

Personal Area Networks (PAN) such as Bluetooth [1] or ZigBee [13] have their own protocol stack defining their own communication systems. For instance, the traditional *Bluetooth Protocol Stack* can be classified in 8 layers that can also be mapped in the OSI model. Newer versions of Bluetooth added a new layer called *Protocol Adaptation Layer (PAL)*, that allows one to combine both *Bluetooth Protocol Stack* and the *TCP/IP* in order to take advantage of these communications while using Bluetooth.

Figure 2.1 shows a comparison of the three models described previously, and the mapping that can be done between them.

Although due to the project constraints most of the work developed in this thesis is at the application level, in order to take full advantage of these systems it is important to know all the layers and how the data travels through them. To finalize, it is important to note that all the work in this thesis is OSI compliant.

2.2 Ad-Hoc Networks

The name Ad-Hoc comes from the Latin and means “for this”. As the name suggests, Ad-Hoc Networks are networks created for a specific purpose. These networks are decentralized, meaning that they do not rely on a already existing infrastructure such as access points or routers.

In Ad-Hoc Networks each node participates in routing the data to other nodes, managing the routing and to forwarding of packets dynamically. Many techniques are used for these purposes, such as controlled flooding or routing tables.

Typically, every device participating in an ad-hoc network has the same status, and it is free to associate itself with any other ad-hoc network in its link range.

There are many kinds of ad-hoc networks with distinct names. For example, the Bluetooth can form an ad-hoc network called Scatternet. A Scatternet is composed of many smaller groups called piconets. Each piconet is formed by a master and a group of up to seven slaves, and to form the larger scatternet, each piconet is linked to another by sharing a common slave node, or linking one slave node to two different master nodes.

2.2.1 Wireless MESH Networks

The idea behind wireless Mesh networks is to form a network of access points that not only assure a wide coverage of signal, as well as a very reliable network. This is because one can have one access point communicating with several other forming alternative communication paths.

This type of networks are often composed by mesh clients, usually mesh routers and gateways. The mesh clients are often laptops, cell phones and other wireless devices while the mesh routers forward traffic to and from the gateways which may, but need not, connect to the Internet. Also, it is important to notice, that in order to communicate within a mesh network, several wireless technologies can be used together, such is the case of 802.11 [28], 802.15 [29], 802.16 [30] and even cellular technologies.

A mesh network can also be formed by wireless nodes that are static, e.g. mounted on lamp posts. This is the case of most of the current existing mesh networks. That said, there are currently very few implementations of “mesh networks” that are true MANETs, that is mobile in nature.

2.2.2 Mobile Ad-hoc Networks (MANET)

Mobile Ad Hoc Networks are a special kind of ad-hoc networks, and as such, are capable of self-configuration, self-healing, self-organization and self-discovering.

Unlike MESH networks, where the mobility is usually reduced, inexistent or there are fixed mesh routers, in a MANET, every device is free to move independently in any direction. The node mobility is one of the main challenges in these networks, because the network has to be capable of regenerating whenever a node leaves or changes its linked devices, which can occur frequently.

It is also possible to identify specific sub groups of networks inside a MANET. The Smart Phone Ad-hoc Networks (SPAN), use the existing hardware in commercially available smartphones, such as Bluetooth and Wi-Fi to, for example, create a peer-to-peer (P2P) network without relying on the cellular carrier network, access points or any other traditional network infrastructure. These kind of networks used to rely mostly on the ad-hoc capabilities of the IEEE 802.11 protocol, but the recent *Wi-Fi Direct* opened even more possibilities for these networks.

Another common type of MANET are the Vehicular Ad-hoc Networks (VANET), that are used for communication among vehicles and roadside equipment. This kind of networks can help drivers, and prevent collisions, accidents, and other possible hazards.

2.3 Ad-Hoc Network Routing Protocols

Since the appearance of ad-hoc networks, many protocols have been developed and studied in order to maintain the consistency of these networks, without having any kind of centralized management. Currently, there are dozens of protocols specially designed to handle data routing in these networks. These can be classified in three main categories: proactive, reactive or hybrid. There are also some hierarchical protocols, but most of them can be classified in one of the previous categories.

Each type of protocols has been proven to be efficient in a different kind of ad-hoc network, but there is still no consensus on which protocol is better for general use.

2.3.1 Proactive Routing Protocols

Proactive routing protocols use a “table driven” routing technique. To keep the tables updated, continuous updates are made by all the nodes, so that the network mapping is always updated.

By using proactive routing protocols to communicate, when a node α wants to communicate with another node β in the network, it just needs to check its own routing table to find the route, like wired protocols usually do. To efficiently find the best route, every node uses some sort of shortest path or optimal path protocols.

The obvious advantage of proactive protocols is that every route is calculated instantaneously, reducing possible delays that would appear from a route discovery. Since the routing tables on each node need to be updated in order to have a good functioning network, these protocols are usually better on stable (low mobility) networks.

A disadvantage of these protocols is the need for table maintenance, with significant costs involved. This operation frequently requires broadcasting updates to a neighborhood, or even the whole network, hindering scalability.

There are many approaches that rely on routing tables, being the most famous (and most common), the Optimized Link State Routing Protocol [9] (OLSR). Destination-Sequenced Distance Vector [38] (DSDV) is also a very known and studied proactive routing protocol. More recent proactive protocols such as Babel [21] or Better Approach To Mobile Ad-hoc Networking (B.A.T.M.A.N.) [35], promise to improve upon the older protocols, and introduce interesting approaches to the proactive routing problem. However, since they are not very used on current systems, we will not describe them in detail.

2.3.1.1 DSDV

DSDV was one of the first proactive routing protocols created, and the basic idea behind it is to make every node a specialized router, which periodically advertises its routing tables to other nodes in the network. This protocol is based on the Bellman-Ford algorithm, and was introduced in 1994.

The routing tables in DSDV store the list of all available destinations, and the number of hops to each, and a sequence number which is originated in the destination node. To keep the routing consistency, each node periodically transmits updates, and if there is significant new information, the update is made immediately. This algorithm does not assume any time synchronization between nodes, and makes no assumption about the phase relationship of the update periods.

The routing information is advertised by broadcasting or multicasting the packets that are transmitted periodically and incrementally. Data about the length of time between the first arrival and the best route for each particular destination is also kept. This data is used to make a decision on whether to delay advertising routes which are about to change soon.

The updates required by DSDV consist on the current neighbors, and the routing table of each node. Given that the routing table may change fairly dynamically over time, the advertisement must be made often enough to ensure that every node can almost always locate any other node in the network.

Although DSDV is not very used nowadays, it was the inspiration for the most used routing protocols being used in modern ad-hoc networks, such as OLSR and Ad hoc On-Demand Distance Vector (AODV) [37].

2.3.1.2 OLSR

OLSR is one of the most used proactive routing protocols in MANETs nowadays. Being a proactive protocol, as it happens with DSDV, some nodes maintain routing tables that are used to calculate the best route for messages.

Unlike DSDV where every node kept a routing list and has to advertise periodically its neighbor list to the network, which lead to a high network flooding, OSLR relies on *Multipoint Relays (MPR)* which provide an efficient mechanism for flooding control of traffic by reducing the number of transmissions required.

Each node in a network using OLSR routing chooses from its neighbor list a set of nodes to become a MPR. To be chosen as a MPR the connection between the node that made the choice and the MPR has to be "symmetrical", in order words, the MPR has to be able to communicate with the node, and vice-versa.

The nodes selected as MPR are the ones responsible of declaring the link-state to the network. In their control messages, they have to declare their selectors, thus allowing the other nodes to calculate the shortest routes to every node in the network. This control messages have to send periodically to the network, in order to keep the tables updated, facilitating the efficient flooding control of messages in the network.

This protocol has been developed to work independently from other protocols, so the OLSR does not make any assumptions on the underlying link layer.

2.3.2 Reactive Routing Protocols

Reactive routing protocols provide routes “on demand”. This kind of protocol floods the MANET with a route request whenever a node needs to deliver a message in the network. When the request first reaches the desired communication destination, the route path is made by using the reverse path.

The main advantages of this kind of protocols is that there is no need for route maintenance, which brings a low routing overhead, and is known to be good when the networks are unstable (such is the case of MANETs).

On the other hand, the delay for acquiring a route on larger networks is bigger than on proactive protocols since every route is calculated dynamically on demand. Because of this dynamic route calculation behavior, the complexity of these protocols increases. There is also the possibility of excessive flooding, that may lead to network clogging.

Finally, there are many different protocols that exploit the basics of reactive routing protocols while also try to increase their efficiency. The most famous is the AODV, but the Dynamic Source Routing [32] (DSR) protocol is also widely used in MANETs.

2.3.2.1 AODV

The AODV is one of the most known reactive (on-demand) routing protocols used in MANETs or other Wireless Ad-Hoc Networks and was developed in a joint effort of Nokia, University of California at Santa Barbara and University of Cincinnati.

Although this protocol is purely reactive, it uses some characteristics common to proactive protocols. Routes are established on-demand when they are needed, however once the route is established it is maintained while needed, and since the routes are only calculated when needed, the network remains silent until one node wants to establish a connection.

In order to build a route, AODV uses a *route request / route reply* approach, so when a *Node* α wishes to find a route, it will send a *route request* broadcast message to all its neighbors containing its IP address, current sequence number and broadcast ID and also the most recent sequence number for the route *Node* α is aware. Upon receiving this message, any node other than *Node* α will either send a *route reply* message back to the source if it is the destination node or has sequence number greater or equal do the one in the message. In any other case, the node will rebroadcast the message to its neighbors keeping track of the *route request* source IP and broadcast ID in order to discard repeated requests.

As the *route reply* propagates back in the network, nodes set forward pointers to the destination. Once the source node receives the *route reply* message, it will begin sending packets to the destination. If later the source receives another *route reply* message containing an higher sequence number or lower hop count, it may update its routing information and start using a better route.

The main disadvantage of this protocol is the fact that intermediate nodes can lead to inconsistent routes, multiple route reply messages to a single route request can lead to heavy control overhead, and the route calculation delay is higher than on proactive protocols.

2.3.2.2 DSR

The Dynamic Source Routing (DSR) is another well known reactive protocol and in principle it is similar to the AODV although it uses source routing instead of relying on a routing table in each intermediate node.

As with other reactive protocols, DSR was developed with the objective of reducing the bandwidth consumed by control packets by eliminating the need to keep routing tables updated, calculating this routes on-demand by broadcasting a route request message when needed, and caching the routes found for a period of time.

This protocol works in a simpler way than AODV, because when a node α wants to find a route to a node β it simply broadcasts a *connection request* message to its neighbors containing a sequence number. Upon receiving a message of this type, each node γ will check the sequence number to verify if the message is new the reverse path. The route calculated will be kept in node α and then be used to communicate from this node to node β.

As promised, this protocol avoids the periodically flood of the network used in proactive protocols to keep the tables updated. The main disadvantage is that the route maintenance mechanism does not repair broken links. Old route cache can also lead to inconsistencies during the route reconstruction phase.

2.3.3 Hybrid Routing Protocols

Hybrid protocols were created in order to take the best of both proactive and reactive routing protocols. Most of the protocols that implement an hybrid routing protocol start by using a proactive approach to map the initial network state, and then use a dynamic, or reactive protocol to flood the network and find new paths.

There are some implementations of hybrid routing protocols, such as Zone Routing Protocol [25] (ZRP), Sharp: Hybrid Adaptive Routing Protocol [39] (SHARP), and Hybrid Ad Hoc Routing Protocol [36] (HARP) but the results obtained from their study are not very enlightening yet, but nevertheless it is important to understand the idea behind these protocols and so a further explanation on the ZRP protocol is given next.

The ZRP is an hybrid protocol that uses both proactive and reactive approaches when sending information over the network, in order to speed up delivery and reduce processing overhead. To achieve this, ZRP chooses the most efficient protocol to use throughout the route.

A node belongs to a zone that is delimited by all nodes that are within a certain radius, which is determined by a given maximum number of hops.

Each node is required to know the topology of the network within its routing zone and nodes are updated about topological changes within their zone.

Keeping the topology information of each node zone updated, allows ZRP to use a proactive protocol to find routes of nodes within a certain zone, without having the high bandwidth demand of traditional approaches. If on the other hand, the *destination node* is not contained within the *requesting node* zone, a reactive protocol is used in order to find which zone contains the desired *destination node*. Once that zone is found, a proactive protocol is again used to find the final route between these two nodes.

With this strategy, packets with destinations within the same zone are delivered immediately. If the destination is outside the zone, the overhead of checking every routing table along the way is avoided using the reactive protocol to find the destination zone. Thus, ZRP reduces control overhead for longer routes that a proactive protocol would imply while at the same avoiding the delays caused by the route-discovery phase of the reactive protocols.

2.3.4 Energy Aware Routing Protocols

Energy efficiency is very a important aspect in most of the mobile ad-hoc networks. Mobile devices, e.g., wireless sensors, mobile phones, have very strict energy limitations. It was with this kind of device in mind that some protocols were developed to improve energy efficiency. This kind of routing protocols try to reduce energy consumption by either minimizing the active communication energy required to transmit or receive packets or the inactive energy consumed when a mobile node stays idle but listens to the wireless medium for any possible communication.

There are several energy aware proposed protocols. Some of these protocols are new approaches, while others are modifications of existing ones. For instance, Gupta et al [24] propose modifications to the AODV protocol. In this work they use new routing cost metrics, that work in function of the remaining battery level. The idea of the cost metric is to be able to route around where the battery is running low. They also propose ways to switch on or off the radio interfaces dynamically for periods of time while the nodes are idle.

Another example is described on [20]. In this approach they modify the DSR, to turn it into a multi-path and energy-aware on demand source routing (MEA-DSR). This protocol exploits route diversity and information about batteries-energy levels for balancing energy consumption between mobile nodes.

2.4 Current MANET and MESH projects

Over the last few years some projects have been developed that take advantage of MESH and MANET technologies. Most of these projects, are either proof of concept implementations in different devices and operating systems, or try to tackle very specific problems and often require specific hardware or kernel code. This means, that although the capabilities of ad-hoc networks are immense, most of the projects existing currently do not yet take full advantage of these networks.

Despite these limitations, it is very interesting to study and test these systems, as they help to better understand the challenges of implementing a MANET or a MESH, while at the same time providing intelligent approaches to these same challenges. For these reasons, in the context of this chapter, some of the most famous were studied, and their goals and features are described next.

2.4.1 Open Garden & Firechat

Open Garden & Firechat are two applications developed by the Open Garden [5] company. Both applications form MESH networks, using Bluetooth, mobile network (3G/4G) and Wi-Fi to form these networks, and work out-of-the-box in Android, Windows and Mac Os X operating systems. These applications allow users to share their Internet connection among several devices connected in the MESH. They promise to its users increased Internet coverage on areas where its access can be limited, such as public parks, beaches, etc.

Although this project is closed source, which makes it less clear on the techniques used to form these networks, and how communication is achieved between devices, the Open Garden company offers an API that allows developers to use their MESH generation, and communication infrastructure on their own applications.

Firechat is a chat room application, using mobile ad-hoc networks that allows its users to create conversation rooms, to communicate with other people using the same application, without having to be connected to the Internet. This application is available on both Android and Apple iOS mobile operating systems, and uses the Bluetooth and Wi-Fi to form the MESH. The code is also closed source, and although this application was developed to be used in concerts and other crowded events, it gained much attention from the media during the Hong Kong manifestations, where protesters were able to communicate with each other even thought there was no mobile network or Wi-Fi available.

2.4.2 The SPAN Project

The SPAN (Smartphone Ad-Hoc Network) Project [6], is a framework and proof-of-concept implementation of a functional MANET for the Android operating system.

All the code of this project is open source and is available on GitHub.

This project uses Wi-Fi Ad-hoc communications, and requires users to install a customized kernel, making its compatibility very limited to supported devices, and requiring users to root their devices.

There are three applications developed by the SPAN Project team. The MANET Manager, allows users to configure and manage a MANET. Using this application on several devices within ∼500 ft (∼152.4 m) of range will allow for them to recognize each other automatically, form the network and communicate. The team also developed a visual representation of the formed network, with its MANET Visualizer application. Finally, there is a voice chat application, called MANET Voice Chat, that allows users to talk with each other in a MANET [41].

2.4.3 Serval MESH

The Serval MESH is an open source Android application developed by the Server Project team, and allows users to form a MESH network that can be used to make voice calls, send text messages or share Internet Access.

This project requires root access, to install a modified wireless driver and enable the Wi-Fi Ad-Hoc mode, and is known to work without problems in a small range of devices and with some minor problems in some others, but all in all, the amount of compatible devices is very small.

This software supports multi-hop voice communication but it is based on “best effort”, and it is not intended to replace conventional telephony as its not reliable in terms of performance. The software also allows file sharing over the MESH, using the Rhizome file distribution service, regardless of the size, content or intended recipients.

This project has been initially funded by many foundations and individuals, such as New America Foundation’s Open Technology Institute, the Shuttleworth Foundation, and Nlnet Foundation. Although there are some known issues, that need to be fixed, the latest release is from October 2014 (about 8 months old), and the current development status of the project is unknown.

2.4.4 NYC Mesh and Guifi

The NYC (New York City) Mesh [4], is a community project that aims to create a free, resilient, stand-alone communication system that serves for daily use and also for emergencies by running software that helps the community with hyper-local maps and events.

Every New Yorker is allowed to join and contribute to this MESH network, by installing a router that connects to other routers in range. With every node that joins this network the service gets better. The routers used by this network run OpenWrt with a package developed by qMp, and use the BMX6 Mesh protocol. Their software also supports tunneling via regular Internet to form the Mesh when the router is out of range. The hardware can either be a small home router, to be used inside a home, or a directional router antenna that can be mounted outside.

In Spain, a similar project called Guifi [2] aims to create a Mesh network available all over the country. This network allows users to browse the Intranet and share the Internet connection. Currently this project has more than 28000 nodes and began in the early 2000s as a way to bring Internet to rural Catalonia where ISPs were not offering connections. Although this project is several years old and the Internet coverage has increased, Guifi is still being used to bring Internet access to some villages in Spain, and its numbers continues to increase, making it one of the largest MESH networks in the world.

Chapter 3

Simulation Frameworks, Environments and Techniques

In this chapter we describe and analyze the state-of-the-art simulation frameworks, environments and techniques, in order to evaluate their adequacy to our research problem.

In order to understand the choice made, this chapter will begin by introducing the concept of simulation, why it is important and the different simulation techniques available. Then, some of the best known simulation tools and frameworks and techniques are described and analyzed, followed by a side-by-side comparison of these same tools. In the last section, the chosen simulator will be analyzed, discussed, and its selection will be justified.

3.1 What is Simulation and why it is important

Ever since the first networks were developed in the 1980s, network simulators have been developed in order to allow researchers and companies to test their models and ideas, without the hassle and cost of performing real implementations of these systems.

But simulation techniques have been used for a long time in order to test and study ideas and models, and make development easier. Simulators try to recreate the environment using theoretical models, such as physics equations, mathematical formulas or the behavior of a machine.

Network simulators must allow parametrization in order to test different scenarios with respect to variables like the medium, technologies, protocols and mobility. Most are capable of simulating the various layers of the OSI stack, from the physical-layer to the application-layer.

There are plenty of simulation tools and techniques available, and also many different types of simulation, which will be described next.

3.1.0.1 Why simulate?

In order to study and analyze the behavior of network algorithms on MANETs, simulation framework are a must. Very often, the most interesting scenarios and problems scale are incompatible with actual physical experimentation with devices (e.g., too many devices required, complex physical deployment).

3.1.0.2 Types of simulation

There are three main kinds of simulators: real time, continuous and discrete.

Real time simulators are good for cases where the event time is relevant, such as in video games or some controller tuning. This kind of simulator may also be useful to perform time sensitive simulation. For example, if one wants to use virtual machines or real devices, connected between themselves over a simulated network in order to exchange time sensitive data. In cases like this, if the simulation does not happen in real-time, there might be problems with the software running on the devices. This type of simulation is usually less scalable than discrete simulation, since all events have to happen in real-time, and this can be computationally very expensive.

Continuous simulation is a computer model of a physical system which continuously tracks system response according to a set of equations. These models are mostly used in physics, to simulate phenomena such as rocket trajectories, robotics, etc.

Finally, the last type of simulation, the one that is really relevant for the work produced in this dissertation, is discrete event simulation. In this type of simulation, models are created to represent a sequence of events in time. Each event occurs at a particular instant, and it marks a change in the system. The main advantage of this kind of simulation is that when an event happens, we skip right to the next one, without having to wait between them. This system of events, differentiates this simulation from the continuous.

Since real time is not a mandatory requirement to study the behavior of the networks, the most adequate simulation model for the work performed in this dissertation should be discrete, although a real time simulation environment was also studied.

3.2 Simulation Frameworks

Next we describe and analyze several network simulation frameworks in order to decide which one is the most adequate for our research.

Some simulators are fully discrete, such as NS-2 [40], others can behave as both discrete and real-time, NS-3 [26] and OMNeT++ [42], and finally there are some that are only real-time simulation tools, such as Mac802.11 and WDSim [12].

3.2.1 OMNeT++

OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, mainly used to build network simulations. The first version of this software was released in 1997, and many independent groups have been developing components for this tool.

Although not intended for network simulation, the OMNeT++ framework has been progressively extended to do so and is now extensively used by both the scientific community and industry.

The framework is based on the Eclipse IDE, which offers a graphical runtime, and a host of other tools. Its modules are written in C++, and are then assembled into larger components using an higher-level language (NED).

There are plenty of extensions ranging from real-time simulation, network simulation, mobility, and several other things.

The INET Framework, is used in OMNeT++. This framework can be seen as the standard protocol model library. It contains models for the Internet stack, link protocols (both wireless and wired), support for mobility, and some MANET protocols. This framework is maintained by the same team as OMNeT++, and accepts patches and new models developed by the community. This framework is also used as a basis for some other tools, such as OverSim, which implement several peer-to-peer overlay algorithms, or Veins which simulates Inter-Vehicular Communications.

Other frameworks include the MiXiM [33], which is a modeling framework created for mobile and fixed wireless networks, concentrating mainly on the lower levels of the protocol stack, and Castalia [18] that is a simulator for Wireless Sensor Networks (WSN), Body Area Networks (BAN), and in general networks of low power devices.

Having multiple teams working on different packages can be viewed as a downside, since it generates some fragmentation, and it makes it harder to combine multiple modules to simulate some scenarios. Some other features are lacking in this framework, for instance, the ability to simulate some link protocols such as Bluetooth out-of-the-box, support for 802.11z (TDLS) and Wi-Fi Direct.

3.2.2 NS-2

One of the first simulators, the *REAL Network Simulator* was modified to give origin to the *Network Simulator (NS)* which in turn, was updated to the *Network Simulator 2 (NS-2).*

The NS-2 was the most used and published simulator for a long time. The reason for this, was that it was efficient, reliable, open-source and had a very large community that developed many modules to expand the default capabilities.

NS-2 is written in TCL and C++, but most of the simulation scenarios are written in TCL. By default, this simulator is capable of simulating TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

Although NS-2 does not simulate Bluetooth, there is an extension that enables Bluetooth simulation. UCBT implements a full Bluetooth stack, but the last version released only fully supported Bluetooth 1.1.

Parallel execution is also available via a community developed package called Parallel/Distributed NS (PDNS). PDNS was developed by the PADS research group at Georgia Tech, and its main goal was to allow a network simulation to be run in a parallel and distributed fashion, on a network of workstations.

The main drawback with this project, is the fact that it is not updated since 2009. At the moment, most of the models supported and protocols are outdated, and although there are still some publications with it, it is becoming a dead project.

3.2.3 NS-3

NS-3 is a recent open-source discrete-event network simulator developed with the intent to replace NS-2, and targeted primarily for research and educational use. This simulator is among the best available, and is one of the most used network simulators nowadays.

It was intended as a substitute for *NS-2*, and was developed from scratch in both C++ and Python. Most of the models that existed by default in the *NS-2* were also ported to NS-3. Many more models have since been developed specifically for this tool.

The *NS-3* is open source, being developed by a very active community, releasing updates every 2-3 months. At the moment, this simulator is capable by default of simulating Wi-Fi, WiMAX, LTE as IP based wireless networks, Point-to-Point, CSMA and other wired ways, and even non-IP based wireless networks, such as ZigBee (802.15.4).

Regarding the Wi-Fi communications, the *NS-3* offers models for the Wi-Fi physical layers 802.11 a/b/g/n at both 2.4 and 5.0GHz, QoS-Based ECDA 802.11e, plenty of propagation loss and delay models such as constant-rate or Log-Distance, various control rate algorithms such as Aarf or ConstantRate and even support for the 802.11e (MESH).

This simulator is also good to test MANETs, since it supports a large range of mobility and building models, and some MANET routing algorithms, such as AODV or OLSR. The models used for Wi-Fi communications (range, propagation loss, SNR) are also very well known, and are seen by the scientific community as very reliable.

Although NS-3 is a discrete-event simulator, it also features a real-time scheduler, that facilitates a number of "simulation-on-the-loop" use cases for interacting with real systems. It is also possible to use a "tap-bridge" network device, to connect a simulated network to a real application or a virtual machine.

Finally, a "Direct-Code Execution" environment is being developed for the *NS-3*. It allows to run real applications and kernel code, as for example a Unix ping program instead of one developed specifically for the framework, directly on the platform.

Besides all the simulation features offered by the *NS-3* to increase performance and to allow for even larger simulations, it is possible to run this environment using MPI for parallel and distributed execution. All these factors make the *NS-3* one of the most complete solutions available to test network configurations, overlays and applications.

The main disadvantage of the *NS-3* is the fact that there are still some communication protocols that have not been ported, such as Bluetooth, the steep learning curve, the fact that some 802.11 extensions and protocols, such as 802.11z (TDLS) are not implemented and the lack of support for Wi-Fi Direct simulation. This can be surpassed by the fact that being an open-source project, anyone is allowed to modify it in any way desirable, although this may bring consequences, because for a theoretical model to be recognized, it needs a much foundation and testing.

3.2.4 WDSim

WDSim is a framework developed to simulate the Android Wi-Fi Direct API. This framework consists of two separate modules. To simulate the Wi-Fi Direct protocol and Network, there is a console that enables developers to control some properties of the simulation, such as adding devices, establishing neighborhoods and moving devices in and out of reach of each other.

The second component of this framework are virtual machines that run android applications. Each of the virtual machines must have local IP addresses, and allow connections on specific ports. An android API is available to enable the use of the WDSim framework by applications.

Although this simulator looks promising, its implementation, still lacks many components that are important when performing realistic simulations. The Wi-Fi Direct protocol is over simplified, ignoring the group formation steps, there is no propagation loss models implemented, the mobility is very limited allowing only devices to be within or outside of the range and there is no DHCP server on the Group Owner, which means all IPs have to be hard-coded on the simulation set.

3.3 Other Simulation Techniques

Although many tools have been developed to perform network simulations, there are other techniques and tools that can be used to run simulations. The first technique analyzed is a Linux driver specifically developed to simulate the behavior of the 802.11 MAC, which allows the simulation of Wi-Fi connections between applications. The second solution is based on the use of Virtual Machines in order to simulate a network with each VM simulating a mobile device, simulating the *churn* effect and other things.

3.3.1 mac802.11 hw_sim

mac80211_hwsim [45] is a Linux kernel driver that was developed with the simulation of the 802.11 protocol in mind. This module allows users to simulate an arbitrary number of Wi-Fi radios for the mac80211, and was introduced in the Linux kernel in 2008.

The mac80211 is a framework which driver developers can use to write drivers for SoftMAC. The mac80211 supports both station mode (STA) and Access Point mode (AP). This framework can use the 802.11 a/b/g/n physical modes, has support for 802.11s (MESH), wpa_supplicant which allows the use of Wi-Fi Direct, supports QoS.

The simulated radios do not have the limitations of real hardware, so it is easy to generate an arbitrary test setup and always reproduce the same setup for future tests. In addition, since all radio operations are simulated, any channel can be used in tests regardless of regulatory rules.

In order to use this driver to perform simulations, it is necessary to generate network interfaces that make use of it. If one wants to use Wi-Fi Direct, assigning different interfaces for each application and enabling Wi-Fi Direct on the wpa_supplicant will allow to connect these applications among themselves using the virtual network provided by the mac80211_hwsim. If on the other hand, a user wishes to run a traditional AP network, one of the interfaces has to be configured to act as such, and then, the applications, using their own interfaces can connect to the same network via the AP.

Although the original driver did not support mobility, a modified driver has been developed [31] so that mobility and the medium can be simulated.

3.3.2 Virtual Machines

Another way to simulate the behavior of applications in a network, without looking into the communication protocols used, is to use a set of Virtual Machines, connected with each other using a bridged device.

If for example, one wants to simulate a network where different nodes communicate in an ad-hoc way, each Virtual Machine can became one node, running the application to be tested, and the communication can be done using the network adapters created for that purpose. If the *churn* effect and mobility should be taken in account, it is possible to generate mobility trace files using some other simulation tools for the network being simulated, and establish triggers to establish node communication.

This type of simulation can be useful, for instance, to simulate a group of mobile devices that communicate with each other. A similar approach to this simulation was done by the WDSim team, although in that case, there was an external server running to control the network behavior, and the simulated network was limited to an application instead of the whole operating system.

3.4 Simulator Comparison

In order to compare the existing simulation alternatives and be able to select a simulation tool suitable for our research, we selected a set of relevant features that allowed us to classify and to distinguish the tools among themselves.

The feature criteria chosen are the following:

- Source Type - Whether the code is open source or commercial
- Coding Language - Programming Language used to write models and applications
- Development Status - Whether the tool is being updated or the development has stopped
- CPU Usage - Whether the tool is CPU intensive or not
- Memory Usage - Whether the tool consumes high amounts of memory or not
- Performance - How it performs while running simulations
- Mobility Support - Whether it supports mobility or not
- Protocols available - What the amount of relevant protocols available by default
- Community status - Whether the community behind the software is active
- Publication amount - Whether it is a widely used tool in the scientific community
- Energy Models - Whether or not the tool has energy models available
- Parallel Execution - If the tool can perform parallel simulations

The results shown in the table 3.1 were gathered from different sources and from experimental data. Performance, CPU and memory usage data was gathered from the work done by Bilalb et al [17] and by Weingartner et al [44]. The number of publications was compiled from different sources, such as Google Scholar or the different simulation tools homepage.

	OMNeT++	**WDSim**	**mac802.11 hw_sim**	**NS-2**	**NS-3**	**Virtual Machines**
Source Type	**Open**	**Open**	**Open**	**Open**	**Open**	Closed/Open
Coding Language	**C++** NED	**Java**	**C**	**C++** TCL	**C++ Python**	—
Development Status	**Active**	Unknown	**Active**	Stopped	**Active**	**Active**
Performance	**High**	Unknown	**High**	Low	**High**	**High**
CPU Usage	High	High	**Low**	High	High	High
Memory Usage	**Low**	High	**Low**	High	**Low**	High
Mobility Support	**Yes**	Partial	No	**Yes**	**Yes**	Partial
Protocols Available	**Plenty**	Few	Few	**Plenty**	**Plenty**	Few
Community	**V. Active**	L. Active	Active	L. Active	**V. Active**	**V. Active**
Publication Amount	High	Low	Low	**V. High**	High	Low
Energy Models	**Yes**	No	No	**Yes**	**Yes**	No
Parallel Execution	**Yes**	No	No	**Yes**	**Yes**	No

Table 3.1: Comparison of Simulation Tools & Techniques.

3.5 Discussion

Both the techniques of simulation through emulation or the use of a modified 802.11 MAC are interesting approaches if the problem being simulated is not very dependent on the use of different types of communication protocols. These methods, with very little modifications, allow for testing a real application that can be deployed on a real scenario. Since in the Hyrax project we can not exclude the possibility of using protocols other than Wi-Fi, and there is a need to have good theoretical models to test the behavior of the networks, the use of these techniques was not a viable option.

However, in the long run, it may be interesting to combine both virtualization and simulated networks, in order to facilitate the application testing.

The main disadvantage of using a standalone network simulator for this project is the fact that every model designed, every algorithm created has to be implemented on the simulation tool, before being deployed on a real scenario. This can be tiresome and an inconvenience when, in order to test the models and algorithms there is a need to validate the results in small scale scenarios, so they can be later expanded into large scenarios, while having some confidence that the simulated results will mimic the real world ones.

But, since the network is such an important part of the studies performed, and developing a new tool is very much time consuming, one needs to resort to one of the simulators that were discussed.

Analyses of the table 3.1, and the description of the various simulators, suggest that there are two tools that clearly distinguish themselves from the rest. *NS-3* and *OMNeT++* are clearly the most recommended frameworks to perform the rest of the work proposed.

Although both alternatives are very similar, the choice fell into *NS-3*, since for personal preferences Python and C++ are more user friendly than NED and C++, the performance of *NS-3* is slightly better, there is no need to have a graphical user interface. The updates on NS-3 are also very frequent and the tool is easy to expand.

From this point on, all the simulations presented in this dissertation use the *NS-3*, and in every case, all modifications done to the framework and models implemented will be explained and justified.

Chapter 4

Evaluating Wi-Fi Technologies

Nowadays there are plenty of Wi-Fi technologies available, such as Wi-Fi Direct, Ad-Hoc or TDLS. Each of these technologies has characteristics that can be useful to build mobile ad-hoc networks. For the Hyrax project, it is important that the technologies involved provide the best performance, scalability and stability. In this chapter, we study several Wi-Fi technologies for ad-hoc networks and do several experiments to test their performance in the scenario provided by the project. The results obtained from the study described in this chapter also serve to validate the models implemented in the NS-3.

4.1 Experiment Overview

To develop a solution for an ad-hoc on-demand content provider network for sport venues, it is important to study the existing wireless communication technologies, namely Wi-Fi, in their current capabilities, and how they behave in various scenarios. It is important to test different combinations and network densities, to study what are the limitations on the Access Point (AP), how interferences affect the bandwidth and how multiple technologies behave when they are used together.

In this experiment, the objective is to compare all the available Wi-Fi technologies, that can be used in order to get a deeper knowledge of the alternatives available in Android devices as of May 2015.

For our scenario we want to test the available wireless technologies in multiple configurations, and eventually combined with standard wired protocols. For example, we wish to test scenarios in which a network of devices uses Wi-Fi to connect to a content server through AP. Another scenario involves bypassing the AP as much as possible, using TDLS or Wi-Fi Direct to exchange messages between devices.

Given that NS-3 does not yet include support for TDLS and Wi-Fi Direct, to simulate these technologies we had to emulate their behavior and make some assumptions. Experimental data obtained using real devices, together with IEEE drafts and Wi-Fi Alliance white papers provided the required input for the simulation of these technologies.

In order to do these experiments a client/server scenario was designed. This scenario consists on a number of client nodes that download, all at the same time, content from content providers. Each client downloads 10 equal-sized files, in random order. When more than one server is available, the clients choose randomly from which provider to download. This allows for a random behavior and balanced work between all devices.

The data gathered with this experiment consists of the average download time on each device. This approach is shared across all scenarios in order to make the results obtained comparable.

4.2 Technologies and Implementation

The IEEE 802.11 is a set of PHY and MAC specifications for the wireless communication technology known as Wi-Fi. Most of currently available Wi-Fi implementations are based on this specification. There is also a global non-profit association called *Wi-Fi Alliance*® [7] that proposes and defines new standards that can be added to the 802.11 specification.

Wi-Fi is a wireless technology that had its first version released in 1999, and has since suffered several updates until becoming nowadays the most widely used wireless communication technology.

The next subsection describes this upgrade process, specifying the most relevant features added for each new version. Next, we do a review of the Wi-Fi Direct and TDLS, and finally, we describe the process of emulating these technologies in NS-3.

4.2.1 Wi-Fi Specifications

The Wi-Fi standard has suffered several updates [16], revisions and extensions since it was created. Typically Wi-Fi uses radio waves at 2.4GHz or 5GHz to transmit data between devices. Since the first commercial version 802.11a data rates have increased, new modulation methods have been developed and new technologies have appeared. The 802.11n was the first to introduce the High Throughput (HT) and Multiple Input Multiple Output (MIMO). The MIMO support specifies that the device is capable of parallel input and output communications and the HT means that the device supports high speed transfers.

In the most recent standard, the 802.11ac the HT was upgraded to Very High Throughput (VHT), allowing for even faster speeds. This standard also uses MIMO.

Many extensions have also been developed to improve communication QoS and speed. In the context of this work, one of the most relevant was the 802.11z (TDLS) which allows devices in a local network to communicate with each other directly without having to route information through the AP. This extension is supported by most of the communication standards. Table 4.1 presents a more detailed description and comparison of these technologies [28].

	802.11a	**802.11b**	**802.11g**	**802.11n**	**802.11ac**
Data Rate (Mbps)	6, 9, 12, 18, 24, 36, 48, 54	1, 2, 5.5, 11 24, 36, 48, 54	6, 9, 12, 18, 24, 36, 48, 54	Up to 150 (40mhz)	Up to 866.7 (160mhz)
Operating Band (GHz)	5	2.4	2.4	2.4 / 5	5
Bandwidth (MHz)	20	22	20	20 / 40	20 / 40 80 / 160
Channels Number	12/13	11/13/14	14		
Non-Overlapping Channel Number	12/13	3/4	3/4	3/4	3/4
Range Indoor (m)	35	35	38	70	35
Range Outdoor (m)	120	140	140	250	—
(Very) High Throughput	No	No	No	HT	VHT
MIMO Support	No	No	No	Yes	Yes
Compatible with	a	a/b			a/g/n/ac
802.11z Support	No	No	Yes	Yes	Yes
Wi-Fi Direct Support	Yes	No	Yes	Yes	Yes
Modulation Method	OFDM	DSSS	DSSS ERP-OFDM	OFDM	OFDM

Table 4.1: Spec Comparison between Wi-Fi 802.11 a/b/g/n/ac Protocols.

4.2.2 802.11 Protocol Extensions & New Technologies

Wi-Fi Direct [10], released in 2010 and Wi-Fi TDLS [11], released in 2012, are two new Wi-Fi technologies that try to solve distinct problems. Wi-Fi Direct was created as an alternative to Wi-Fi Ad-Hoc, allowing groups of devices to connect in a local network without the need for a physical infrastructure. TDLS serves as a way to reduce overhead in Wireless Local Area Network (WLAN) communications by allowing two devices under the same AP to communicate with each other without having to send the data through the AP. As an extension of the 802.11 protocol, TDLS is also available for Wi-Fi Direct.

Wi-Fi Direct

Wi-Fi Direct (or Wi-Fi P2P) is a new technology developed by the Wi-Fi Alliance to replace the old Wi-Fi ad-hoc mode, which was never much used because of issues, such as power consumption and insecurity.

In this technology, devices have all the traditional Wi-Fi strengths, but are not restricted to a fixed infrastructure, so it is more adequate for mobile platforms while being secure and easy to use. This technology is even compatible with legacy devices that do not officially support it, because a Wi-Fi Direct device can connect to a regular hotspot and vice-versa.

In Wi-Fi Direct devices discover each other and organize themselves automatically in groups formed by one Group Owner and several Group Members. The Group Owner acts as an AP, and is responsible for keeping the group well formed, as well as routing data transfers between nodes as a regular AP.

When two devices want to connect using Wi-Fi Direct, there are several approaches in which they can negotiate the group ownership and establish the group. In the standard approach, devices perform a regular Wi-Fi scan in order to find already existing Wi-Fi networks and Wi-Fi Direct Groups. After this scan, a special discovery algorithm is used in order to find the other P2P devices. Once two P2P devices have found each other, they start the GO Negotiation phase, which is done in a three-way handshake. Once the devices have discovered each other and agreed on the respective roles, they establish a secure communication, and finally use DHCP exchange to set up IP configuration.

The second approach to form a group is called *autonomous*, since a device may choose to create a P2P group autonomously, where it immediately becomes the GO. The other devices can now connect to this new group using traditional scanning mechanisms.

The last approach to form a group is by doing it in a *persistent* way. To do this, during the formation process, P2P devices can declare the group as persistent, by using a flag in the P2P capabilities attribute present in Beacon frames. Using this method, devices forming the group store network credentials and the assigned roles for subsequent re-instantiations of the P2P groups. This way, if during the Discovery phase two devices recognize the formation of a persistent group with the corresponding peer in the past, any of the two P2P devices can use an Invitation Procedure to quickly re-instantiate the group.

The group connections can be seen as one-to-many since one device will act as a gate keeper to invite other devices and determine whether devices requesting to join the group will be allowed. Similarly, the GO acts as an AP, and as such, all data sent from two group members will go through this node, unless a technology like TDLS is used.

Wi-Fi TDLS

The IEEE 802.11z better known as TDLS is an extension to the 802.11 protocol that allows devices to establish direct connections in a local wireless network without having to send data through the AP.

This technology is compatible with the 802.11 a/g/n/ac, and in order to establish a connection the devices can proceed in two different ways. The traditional way, is for a device that wants to establish a TDLS connection with another, to send a message using the AP, asking about that device capabilities and advertising its own. The destination device, upon receiving this message, can either reply back, and then the originating device establishes the connection, or simply establish it itself.

The second, and easiest way, for two devices to connect using TDLS is for one of the them to simply try and establish this connection without sending any information using the AP.

The main advantage of using TDLS is the fact that sending a message through the AP involves the message to be relayed, thus increasing the bandwidth usage, while sending it directly avoids this. Another advantage is the fact that the connection is made directly between devices, which in most of the cases means faster speeds.

Finally, since the TDLS avoids the use of the AP, the communication will be done using the maximum common capabilities of both devices. For example, if the AP only supports 802.11g and both devices wish to communicate with TDLS support 802.11n while the traditional communication is done using the 802.11g standard, the TDLS would be made using the 802.11n standard.

4.2.3 Implementation in NS-3

The Wi-Fi Direct implementation done in NS-3 was based on the description of the protocol done in [19] and the TDLS implementation was based on the description done in the white paper [11]. For the Wi-Fi Direct values obtained from experiments with real devices were also used to instrument the model.

From the paper describing the implementation of Wi-Fi Direct, the behavior of a built group is very similar to the infrastructure mode, but instead of having a dedicated AP, its role is done by the GO. For legacy devices, the compatibility mode used is to turn the device an hotspot, which once again works in a similar fashion as an infrastructure network with an AP.

To emulate the Wi-Fi Direct group in NS-3, the first node created is assigned a Wi-Fi interface with AP mode, so this node will act as a mobile AP that represents the GO. The role of GO is assigned to this node logically in the application layer, and the application implemented on each node knows this role.

Every other node in the network will become a *Group Member*, which means that they will connect to the *Group Owner* as if it was an AP. Each *Group Member* is a *Remote Station* (`STA` in NS-3) connected to the same subnet as the *Group Owner*, thus forming the Wi-Fi Direct. Once again, the role of *Group Member* is saved logically in each node application.

This approach is similar to the one implemented in WDSim, but since it uses the communication models of NS-3, the group behavior is more realistic than the one in that simulator, and is close to the behavior verified on real devices.

Since the group and roles are assigned *a priori*, in order to simulate the group formation process we simply implemented the average delays observed in the real scenario. This means that the group owner negotiation protocol was not implemented neither was the group join protocol. From various experiments done using *Google Nexus 9* and *Google Nexus 7* tablets, we observed that the average group creation time took between 1.5

to 3 seconds. Using these values, when the group is first formed in NS-3, we generate a random delay in that interval, after which the group is officially created. This implementation brings several advantages, such as the simplified process of adapting this delay to new values, if such is needed in the future.

In the implementation done, protocol errors are not simulated. Although it is possible to simulate this factor, such is not necessary since most of these errors happen during the group formation phase, and the delays obtained from the real scenario already accounted for the delays caused by those errors.

The emulation of TDLS is a simpler process than Wi-Fi Direct. The objective of TDLS is to enable device-to-device communication without information being routed through the AP in an infrastructure network or Wi-Fi Direct group. The most appropriated interface available in NS-3 for this purpose is the Wi-Fi Ad-Hoc, since its communications are device-to-device.

To emulate TDLS, on each mobile device, or group member/owner (in case of Wi-Fi Direct) two network devices are placed. The main device uses regular `STA` or `AP` modes (depending on the node role), in order to simulate the infrastructure and the secondary device uses Ad-Hoc mode. In our implementation, both devices use the same Wi-Fi channel.

To establish a TDLS connection, a device first has to create a regular connection to the destination node using the infrastructure and then, using the secondary device, establish a direct connection via ad-hoc to simulate the connection done in TDLS.

4.3 Network Configurations and Test Scenarios

In order to assess the limitations and scalability of Wi-Fi, TDLS and Wi-Fi Direct, we designed a set of test cases scenarios based on four network configurations. Each mobile client makes 10 random file download requests from a set of video files. Files are equally sized, each with 3MBytes. In some configurations, requests will be served by a fixed server positioned in the Internet, on others content is served by mobile servers that are in the neighborhood of the mobile clients. The following scenarios have been considered:

Wi-Fi - Fixed Server + AP + Mobile Clients

Scenario A: All mobile clients connect via Wi-Fi to an access point (AP) and make requests to a fixed server in the Internet. We simulate one test case in which the mobile clients vary from 1 to 32.

Wi-Fi and TDLS: Mobile Server + AP + Mobile Clients

Scenario B: All mobile clients connect via Wi-Fi to an AP and make requests to a mobile server also connected to the AP. Two test cases are simulated. In the first test case we vary clients from 1 to 32. In the second test case, we vary the mobile servers 2, 4, and 8, and do this for 8, 16 and 32 clients. In this test, servers behave only as servers.

Scenario C: All mobile clients connect via Wi-Fi to an AP and communicate directly with each other and also with a mobile server via TDLS. Two test cases are simulated. In the first test case we vary clients from 1 to 32. In the second test case, we vary the mobile servers 2, 4, and 8, and do this for 8, 16 and 32 clients. In this test, servers behave only as servers.

Wi-Fi Direct: GO + Mobile Clients

Scenario D: One node is the Group Owner and acts simultaneously as access point and content mobile server, all other mobile nodes are clients connected to GO via Wi-Fi Direct. We simulate just one test case in which clients vary from 1 to 32.

Wi-Fi Direct and TDLS: GO + Mobile Server + Mobile Clients

Scenario E: One node is the Group Owner and acts as access point, another mobile node is the content mobile server that connects to GO via Wi-Fi Direct, and all other mobile nodes are clients connected to GO via Wi-Fi Direct. We simulate two test cases. The first test case, clients vary from 1 to 32. The second test case, mobile servers vary 2, 4 and 8, and the clients vary from 8, 16 and 32.

Scenario F: One node is the Group Owner and acts as access point, another mobile node is the content mobile server that connects to GO via Wi-Fi Direct, and all other mobile nodes are clients connected to GO via Wi-Fi Direct and communicate directly with each other and also with a mobile server via TDLS. We simulate two test cases. The first test case, clients vary from 1 to 32. The second test case, mobile servers vary 2, 4 and 8, and the clients vary from 8, 16 and 32.

4.4 Results and Discussion

In this section we show the results for each of the scenarios. We compare the results obtained using different technologies, and draw some conclusions on their behavior, possible limiting factors, and their scalability. All results obtained during these experiments used the TCP communication protocol and IEEE 802.11n. We opted for TCP since we are downloading full videos, and not streaming video in real time, and we want to ensure QoS. This is in a way a worst case scenario, as we could use, for example, UDP and implement our own QoS insurance.

Scenario A - Wi-Fi: Server + Access Point + Mobile Clients

Figure 4.1 depicts Scenario A. This scenario approximates closely current real scenarios, for example sports games, in which a traditional Wi-Fi infrastructure allows mobile clients (MC) connected to an AP to request video replays from a fixed server (S) connected to the AP via a Gigabit Ethernet connection. For simplicity of naming we will refer to this scenario as “WiFi: S+AP+MC”.

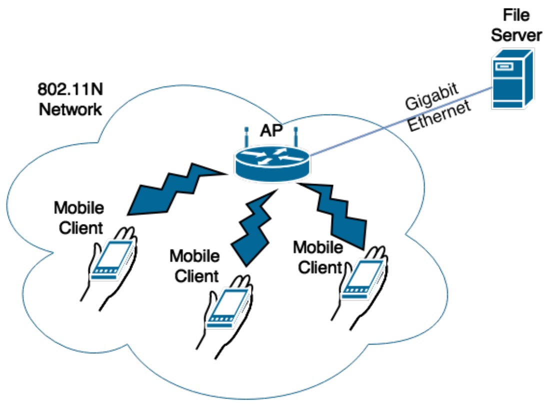


Figure 4.1: Scenario A - Server with AP and Mobile Clients connected via Wi-Fi.

Figure 4.2 shows a graph with the average download time per file, obtained when we vary the number of mobile clients from 1 to 32 with each client making 10 random file download requests.

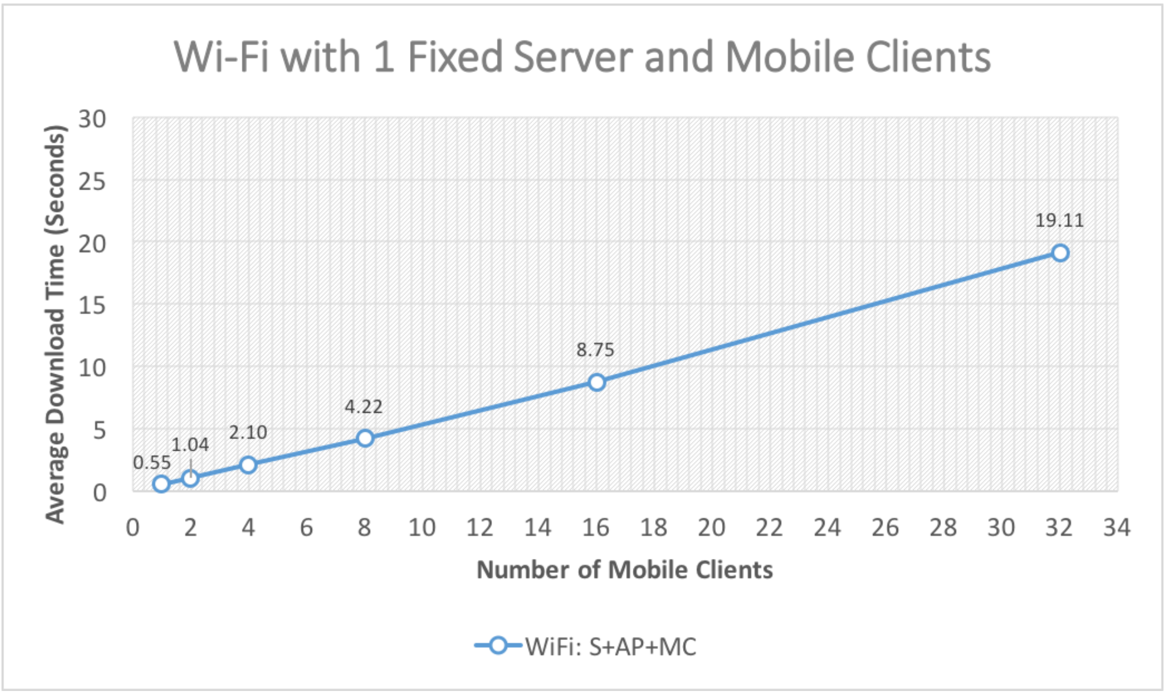


Figure 4.2: WiFi: S+AP+MC (Scenario A) - Average download time by varying Mobile Clients from 1 to 32 using one Fixed Server.

The results show an almost linear tendency for the increase in the average download time when we increase the number of mobile clients up to 32. The line slope increases when moving from 16 to 32 clients, which translates to a degradation of the average download time. This may be a result of the higher density of devices affecting the available bandwidth due to either the communication medium, the AP or the Server. We hope to clarify this with the other scenarios, although the impact of the server should not be significant.

Scenario B - Wi-Fi: Access Point + Mobile Servers + Mobile Clients

Figure 4.3 illustrates the Scenario B in which there is no fixed server or outside Internet connection. All nodes are still connected via Wi-Fi, one acts as a mobile server (MS) and the other as mobile clients (MC). In this scenario, mobile servers do not perform any file requests. For simplicity, we will refer to this scenario as “WiFi: AP+MS+MC”.

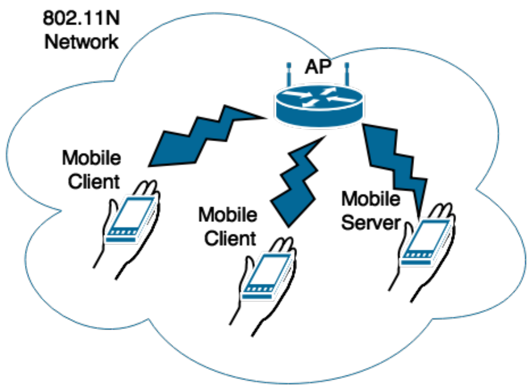


Figure 4.3: Scenario B - Wi-Fi: Mobile Clients and Mobile Servers connected to the AP.

Figure 4.4 shows a graph with the average download time obtained when we vary the number of mobile clients from 1 to 32 with each client making 10 random file requests to a mobile server.

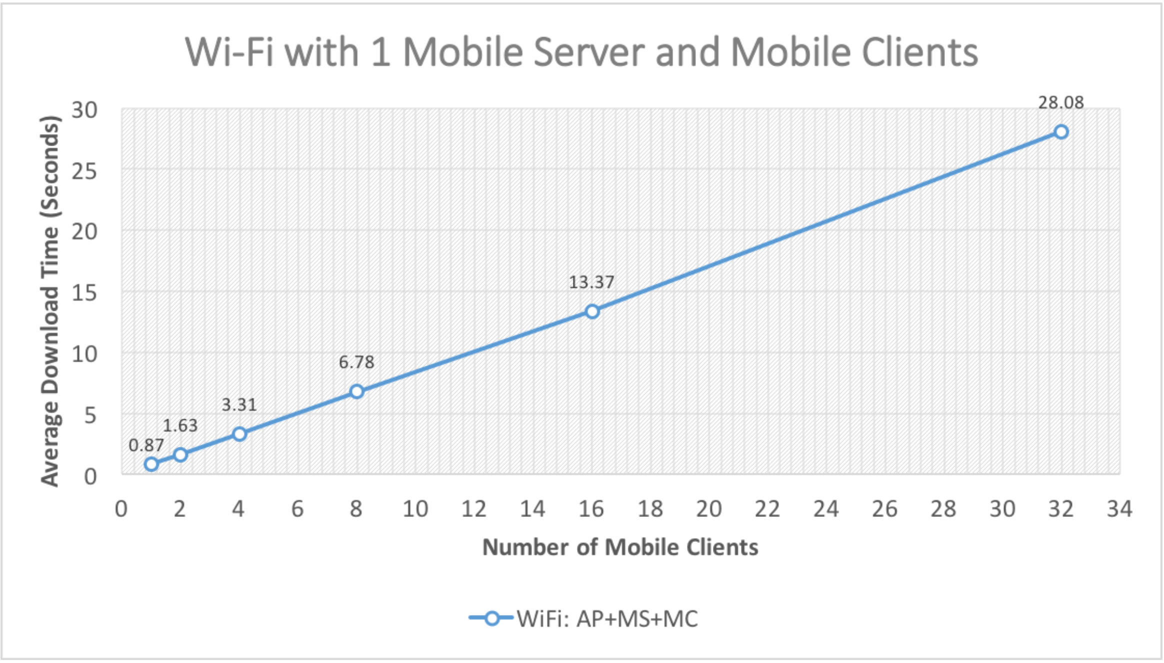


Figure 4.4: WiFi: AP+MS+MC (Scenario B) - Average download time by varying Mobile Clients 1 to 32 with 1 Mobile Server.

The results show again an almost linear tendency for the increase in the average download time when we increase the number of mobile clients up to 32. The increase is slightly higher when moving from 16 to 32 clients. Comparing these results to those of scenario A, Figure 4.2, we see that the average download time degrades by 50% which is understandable as although the whole number of request messages generated by the clients, which go through the AP, is the same as before, but now the forwarding to the server and its response is again via Wi-Fi rather than Gigabit Ethernet.

In this configuration it makes sense for us to assess scalability both when we add more mobile servers and also when we increase the number of mobile clients. The graph in Figure 4.5, compares the results of using multiple mobile servers configurations for 8, 16 and 32 clients. In each of these tests, the mobile servers do not perform download requests.

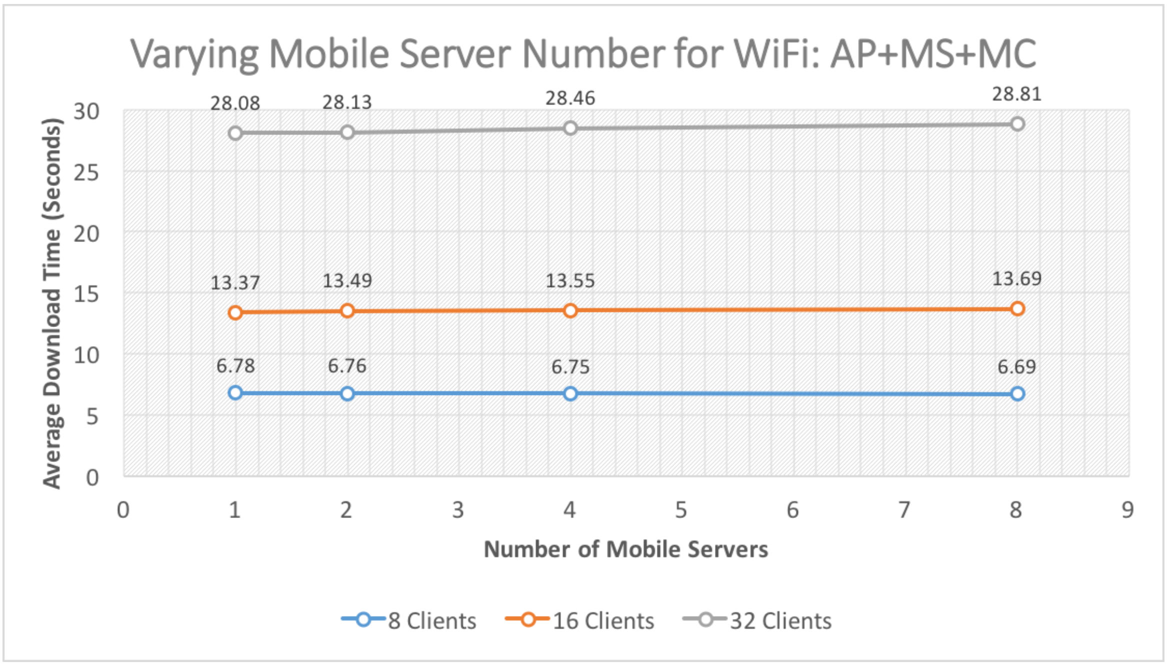


Figure 4.5: WiFi: AP+MS+MC (Scenario B) - Results varying Mobile Servers from 1 to 8, with 8, 16 and 32 Mobile Clients.

Figure 4.5 shows two interesting results. First is that varying the number of mobile servers does not seem to affect the average download time. This is true in the three client configurations. The reason is simple, for a given configuration of clients, say 16, the number of message requests is always the same as is the number of replies coming from the mobile servers independently of their number. Thus the impact on communication is the same, although one would think that concurrency on the servers could have a positive effect. If there is, it is not noticeable on this scenario. The second result is when for a given configuration of mobile servers, say 4, when we double the number of clients the download time also doubles. The increase was expected as the message traffic generated on the AP also doubled and node interference increased. The fact that the increase is not higher, is of interest and hopefully it will be clearer with other scenarios.

Scenario C - Wi-Fi/TDLS: Mobile Servers + Mobile Clients

Figure 4.6 illustrates Scenario C in which all nodes initially connect via Wi-Fi to the AP for discovery and then clients communicate with servers directly via TDLS. In this scenario, mobile servers do not perform any file requests. For simplicity, we will refer to this scenario as “WiFi/TDLS: MS+MC”.

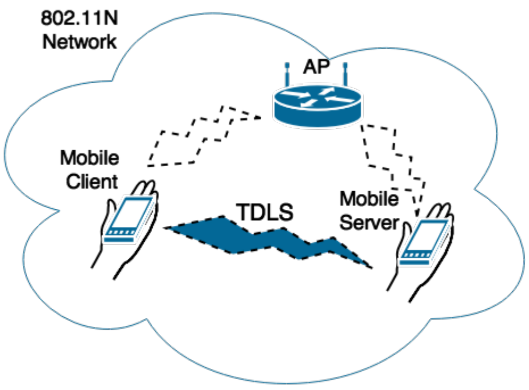


Figure 4.6: Scenario C - Wi-Fi/TDLS: Mobile Clients and Mobile Servers communicate via TDLS.

Scenario C is similar to scenario B, but communication among devices is now with TDLS rather than Wi-Fi. Figure 4.7 shows the average download time obtained for both scenarios (scenario C is shown in blue) when we vary the number of mobile clients from 1 to 32 with each client making 10 random file requests to one mobile server.

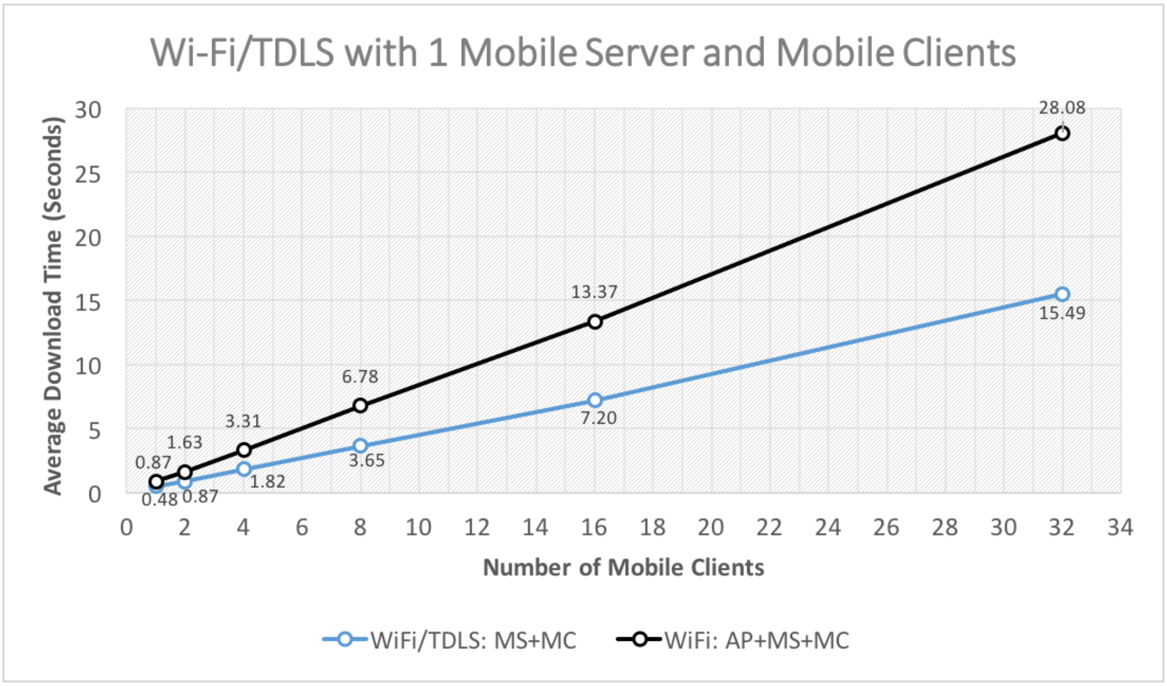


Figure 4.7: WiFi/TDLS: MS+MC (Scenario C) Compared with WiFi: AP+MS+MC - Average download time by varying Mobile Clients 1 to 32 with 1 Mobile Server.

The results obtained show a significant improvement by avoiding the AP with the average download time improving by almost 85% for 32 mobile clients. The reason for this is that using TDLS (scenario C in blue), the number of messages is halved (basically we do 1 hop for direct D2D communication, rather than 2 hops when going through the AP with Wi-Fi. Moreover, we avoid the overload effect from the AP.

Still for scenario C, we assessed scalability both when we add more mobile servers and also when we increase the number of mobile clients. The graph in Figure 4.8, compares the results of using multiple mobile servers configurations for 8, 16 and 32 clients using TDLS for direct communication. In each of these tests, the mobile servers do not perform download requests.

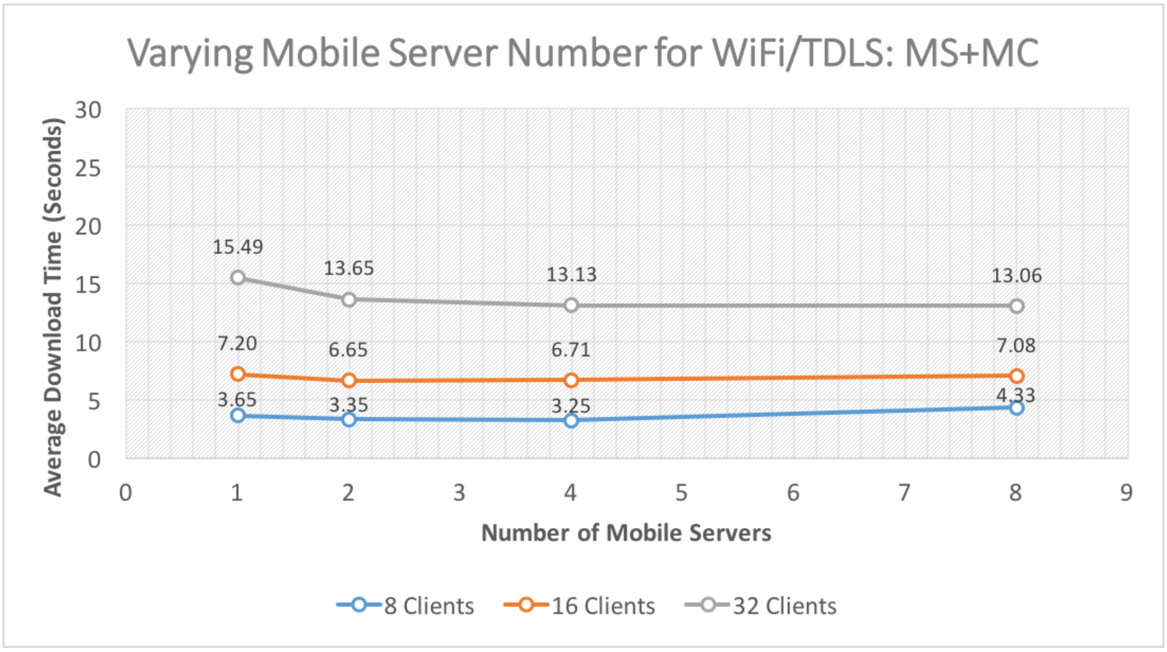


Figure 4.8: WiFi/TDLS: MS+MC (Scenario C) - Results varying Mobile Servers from 1 to 8, with 8, 16 and 32 Mobile Clients.

These results are interesting. In fact, varying the number of mobile servers, specially when density is high, which is the case with 32 clients, does improve the average download time by about 15%, even though the total number of devices increases from 33 to 40. This is the first indication that the medium is not the only limiting factor. The results also show that this is a more scalable solution. In fact, when for a given configuration of mobile servers, say 8, when we double the number of clients the download time increases but is below the double value. This happens while the message traffic generated among devices also doubles and node density increased. These results are very interesting, because although the medium is always limited to the available bandwidth, for larger networks it is possible to reduce the overhead, thus providing a better service.

Scenario D - Wi-Fi Direct: GO is Server + Mobile Clients

Figure 4.9 illustrates Scenario D in which all nodes use WiFi Direct to form a group and the GO is also the server. This means that the GO will operate both functions, as server and as a soft access point. For simplicity, we will refer it as “WiFiD:

GO/S+MC".

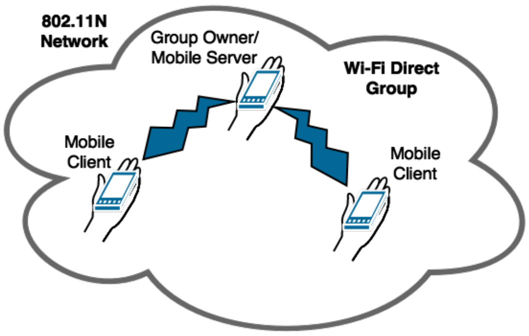


Figure 4.9: Scenario D - GO/Mobile Server with Mobile Clients.

Figure 4.10 shows the average download time obtained for Scenario D when we vary the number of mobile clients from 1 to 32 with each client making 10 random file requests to one mobile server.

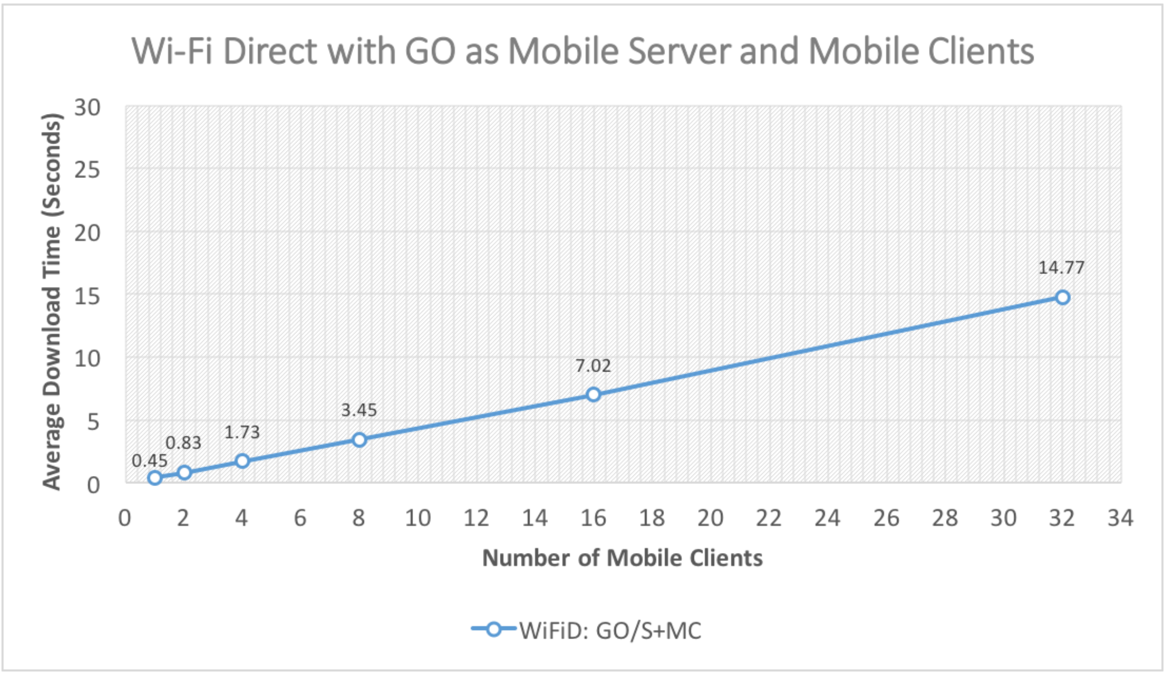


Figure 4.10: WiFiD: GO/S+MC (Scenario D) - Average download time by varying Mobile Clients 1 to 32 with 1 Mobile Server.

The results obtained show that using the GO as a content provider has a positive impact as the average download time is almost 50% lower than what we observed in Scenario B with Wi-Fi and only 20% worse than what we observed in Scenario C ("WiFi/TDLS: MS+MC"). The reasons are similar to those observed with Wi-Fi, a significant improvement by avoiding the AP and by having the GO acting as server, which halved the number of messages to reach the server (basically 1 hop is needed) .

Scenario E - Wi-Fi Direct: GO + Mobile Servers + Mobile Clients

Figure 4.11 illustrates Scenario E in which all nodes use WiFi Direct to form a group controlled by GO that acts just as a router/AP. One node acts as a mobile server while all others are mobile clients making the download requests. For simplicity, we will refer it as "WiFiD: GO+ MS+MC".

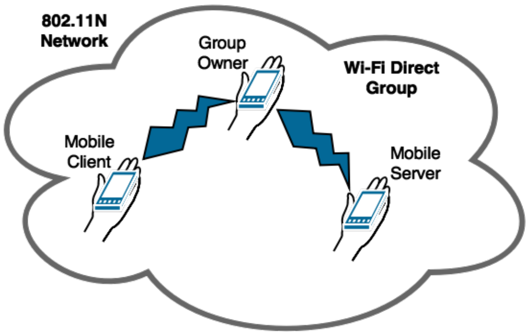


Figure 4.11: Scenario E - GO with Mobile Clients and Mobile Servers.

Figure 4.12 shows the average download time obtained for Scenario E when we vary the number of mobile clients from 1 to 32. The results are much worse compared to Scenarios C and D, but similar to Scenario B. This illustrates the cost of the number of messages going through the GO, in Scenario E, and through the AP in Scenario B, in both cases twice as much needed for Scenarios C ("WiFi/TDLS: MS+MC") and Scenario D ("WiFiD: GO/S+MC").

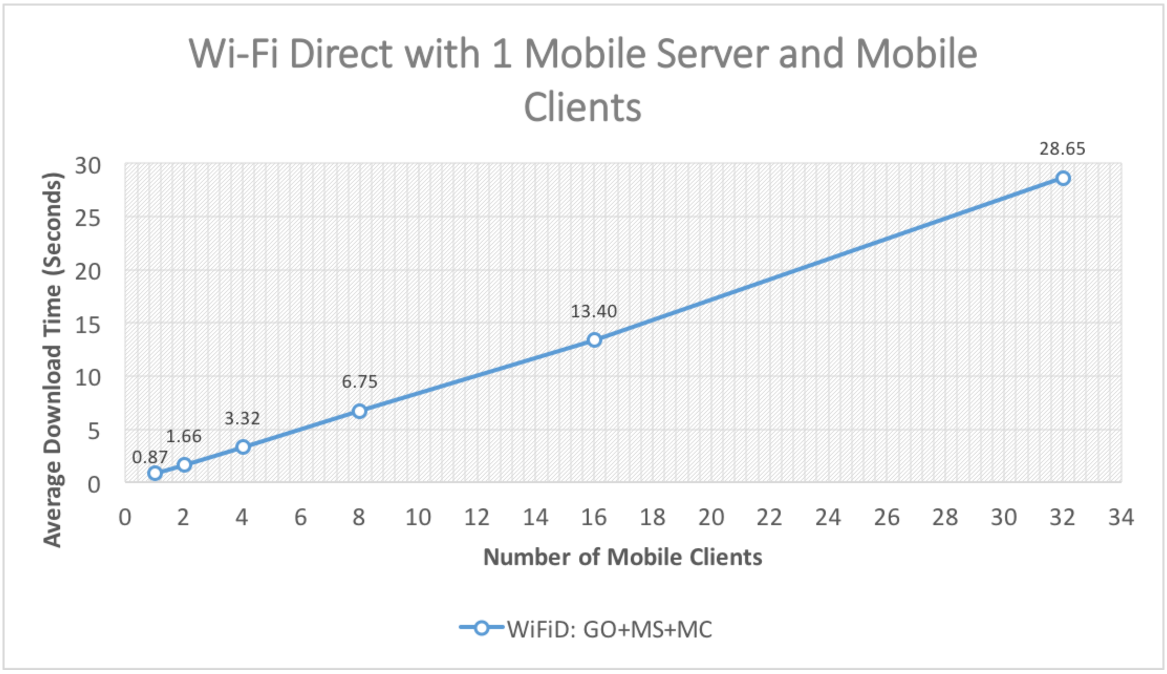


Figure 4.12: WiFiD: GO+MS+MC (Scenario E) - Average download time by varying Mobile Clients 1 to 32 with 1 Mobile Server.

Still for scenario E, Figure 4.13 compares the results of using multiple mobile servers configurations for 8, 16 and 32 clients.

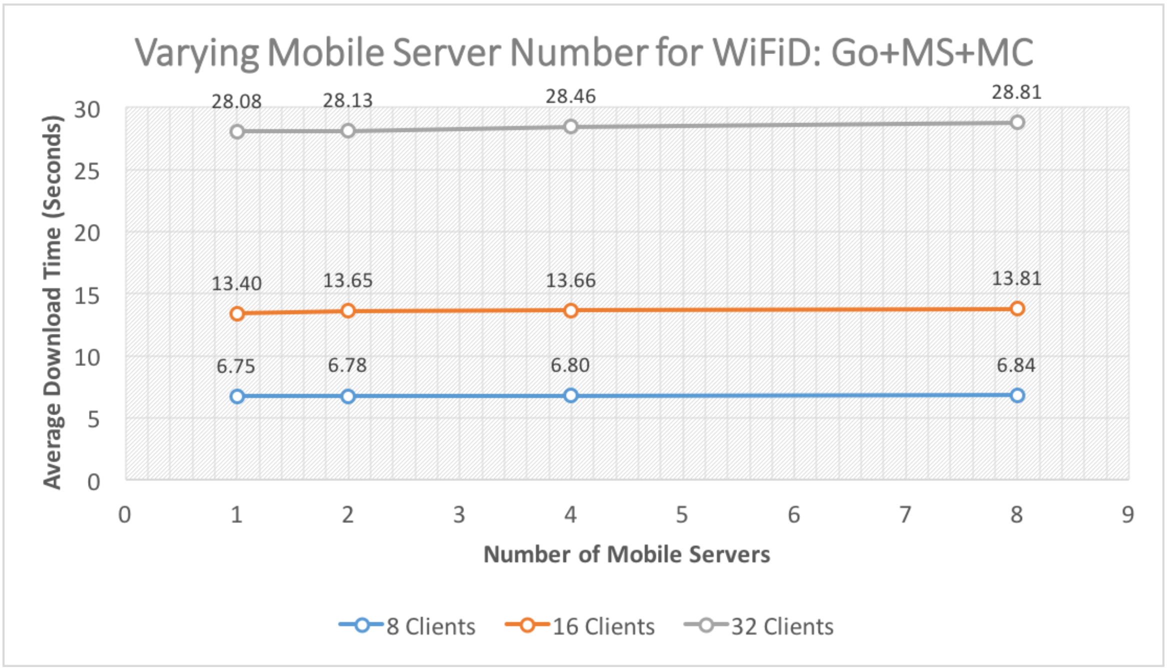


Figure 4.13: WiFiD: GO+MS+MC (Scenario E) - Results varying Mobile Servers from 1 to 8, with 8, 16 and 32 Mobile Clients.

The results in Figure 4.13 show a similar pattern as already observed for other scenarios, namely B. Varying the number of mobile servers does not seem to affect the average download time, which is understandable as the number of message requests is always the same as is the number of replies coming from the mobile servers independently of their number. We can also observe that for a given configuration of mobile servers, say 8, when we double the number of clients the download time also doubles. The increase was expected as the message traffic generated on the GO also doubled and node interference increased.

Scenario F - Wi-Fi Direct/TDLS: GO + Mobile Servers + Mobile Clients

Figure 4.14 illustrates the scenario F, last on our description, which mixes Wi-Fi Direct and TDLS. This scenario is similar to the scenario E ("WiFiD: GO+ MS+MC"), but now instead of using traditional Wi-Fi communications using the GO as a soft-AP, the connection between the mobile clients and mobile servers is made using TDLS. We will refer to it as scenario "WiFiD/TDLS: GO+MS+MC".

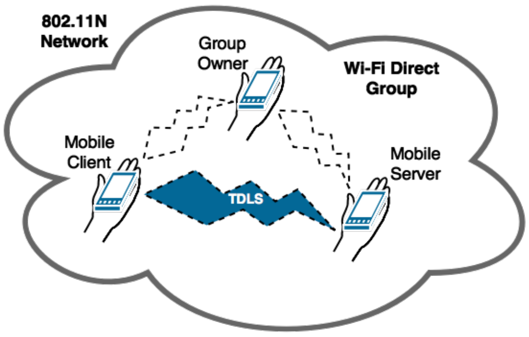


Figure 4.14: Scenario F - GO with Mobile Clients, Mobile Servers and TDLS.

Figure 4.15 shows the average download time obtained for Scenario E (black line) and Scenario F (blue line) when we vary the number of mobile clients from 1 to 32.

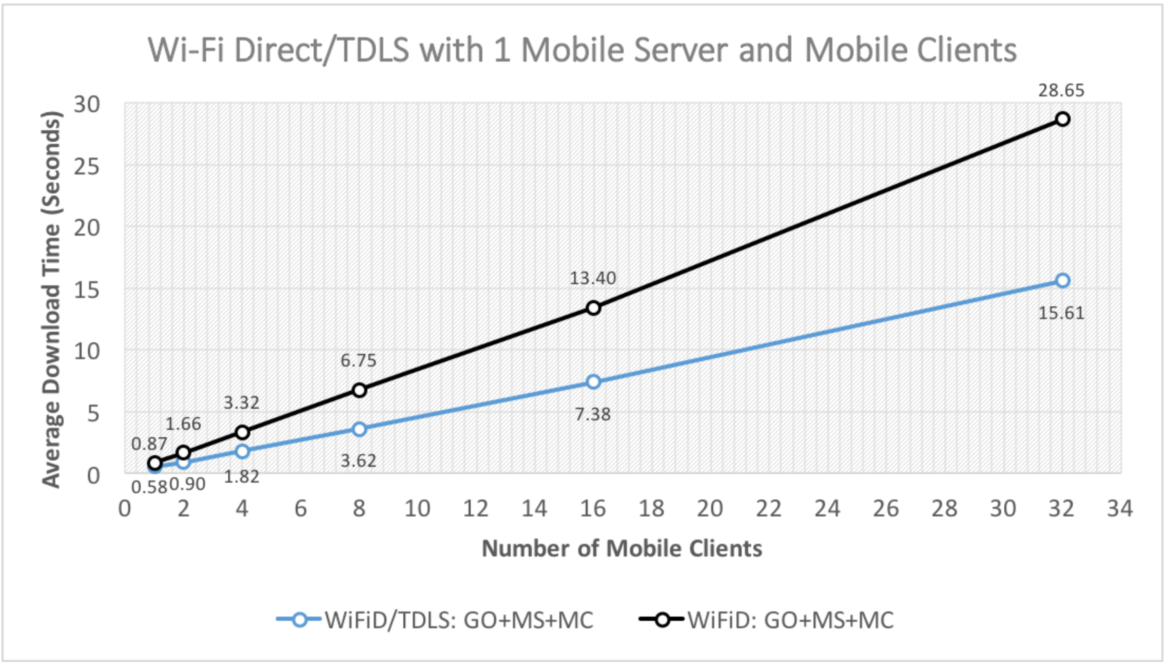


Figure 4.15: Scenario F and Scenario E - Average download time by varying Mobile Clients 1 to 32 with 1 Mobile Server.

The results clearly show the gain in the average download time that can be achieved by allowing direct device to device communication (TDLS) rather than having to go through a GO (or the AP).

Figure 4.16 compares the results of using multiple mobile servers configurations for 8, 16 and 32 clients.

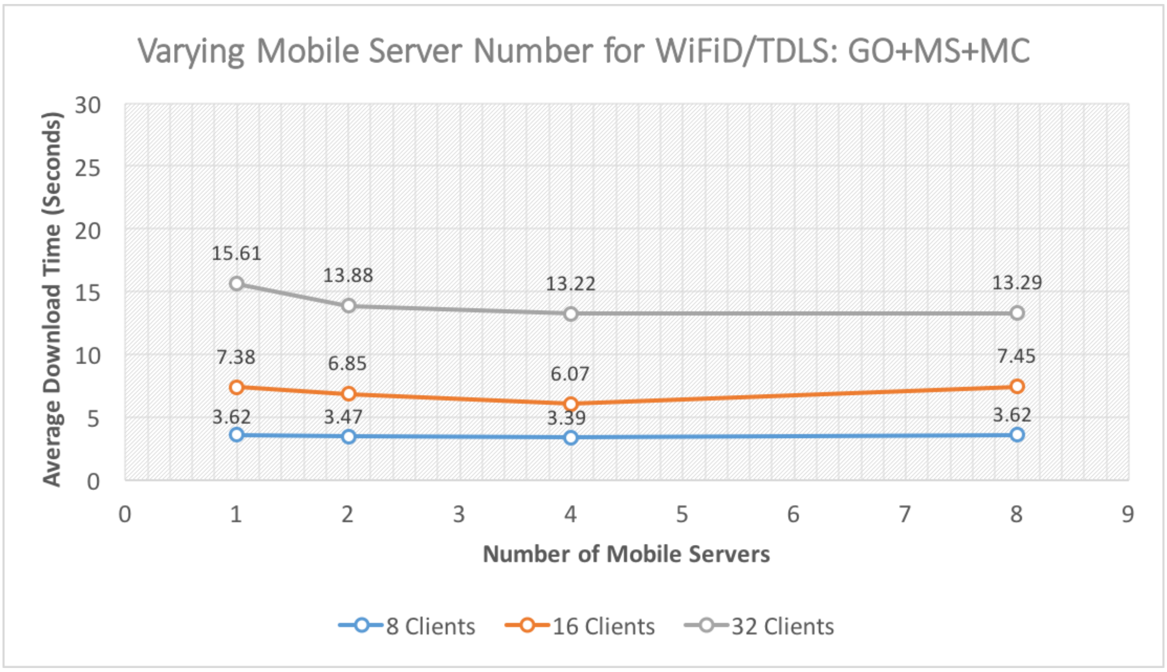


Figure 4.16: WiFiD/TDLS: GO+MS+MC (Scenario F) - Results varying Mobile Servers from 1 to 8, with 8, 16 and 32 Mobile Clients.

The results are similar to those observed in Scenario C ("Wi-Fi/TDLS: MS+MC") with a noticeable decrease of the average download time for both 16 and 32 clients when we use multiple servers. This indicates that by using device to device communication, having multiple servers helps in achieving a better load balance which translates to better average download time. This also indicates that using an AP or GO as a routing node is very limiting and should be avoided if possible.

4.4.1 Scenarios Comparison

In this subsection, comparable scenarios are observed simultaneous so that we can extract a broader picture of the performance of all technologies.

One Mobile Server and Varying Number of Clients

Figure 4.17 displays the average download time for all scenarios, A to F, using just one mobile server and mobile clients varying from 1 to 32. Each client makes 10 random file downloads from the server. In all scenarios, every node starts downloading the files, but otherwise they run until all files have been downloaded without synchronizing with the other nodes.

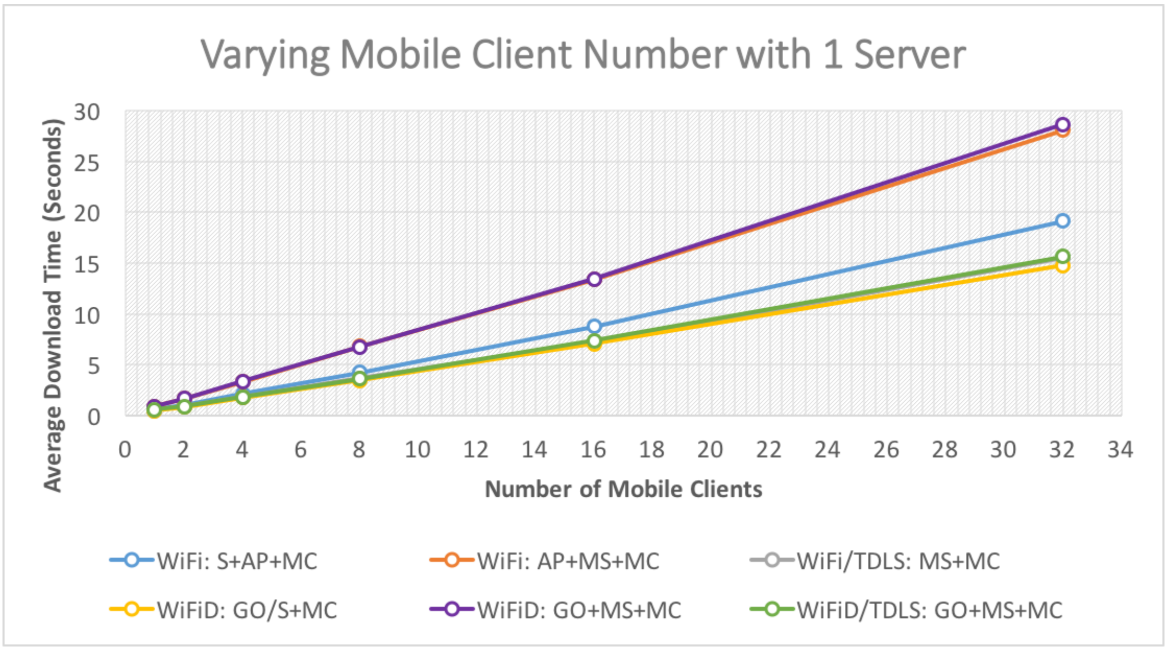


Figure 4.17: Results varying 1-32 Nodes with 1 Server.

The results show that the best performing scenarios correspond to those using TDLS or Wi-Fi Direct with the GO functioning also as server. The reason for this is that both approaches allow for direct communication between devices while all other approaches require messages to travel through a routing device, be it the AP or the GO, generating more hops for reaching the server and to receive its responses. The results for those two scenarios are slightly better than normal access to a server through an AP using Wi-Fi. But the fact that the gain is not much higher can be an indication that most of the delay is due to the effect of the wireless medium.

Although both TDLS and GO as server approaches show good results, it is important to note that it is not possible to increase the number of GO servers within a Wi-Fi Direct group, which is a limitation, thus indicating that TDLS is the best approach to our problem.

Comparing scenarios with 8, 16 and 32 Clients and Varying Numbers of Servers

The following three figures show results that allow us to compare scenarios B, C, E, and F, those relying on mobile servers to deliver content, in the same graph. They show the average download time when we vary the mobile servers in 1, 2, 4 and 8, with clients fixed on 8, 16 and 32. To balance workload among servers, clients select randomly the server from which they request each file.

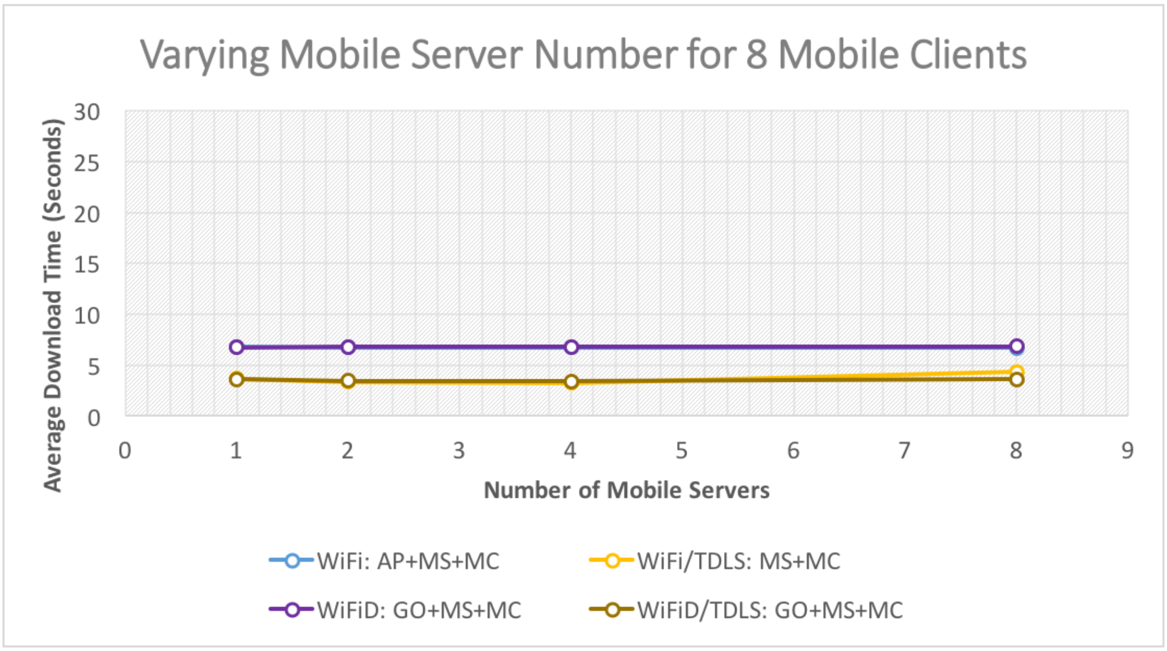


Figure 4.18: Results varying 1-8 Mobile Servers with 8 Mobile Clients.

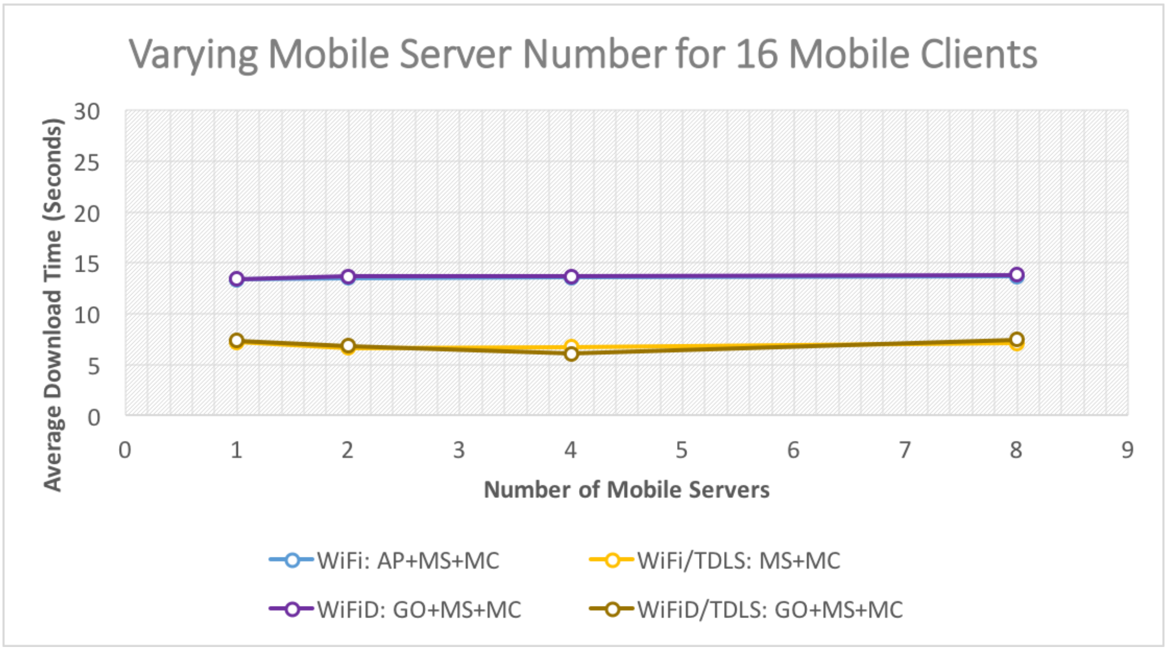


Figure 4.19: Results varying 1-8 Mobile Servers with 16 Mobile Clients.

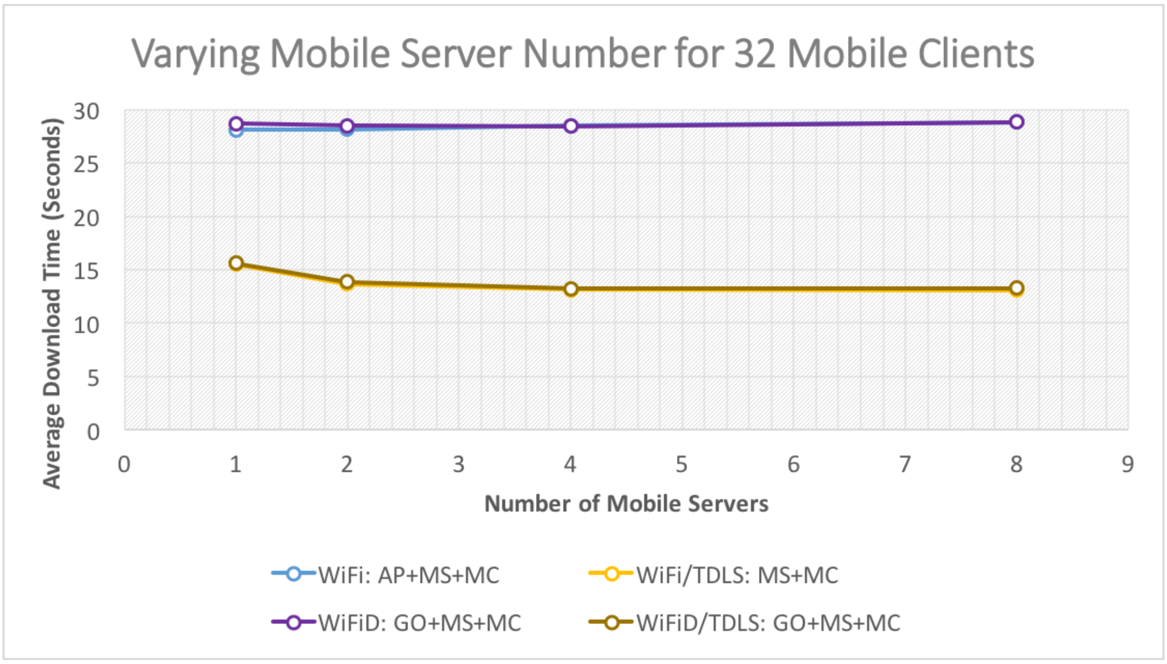


Figure 4.20: Results varying 1-8 Mobile Servers with 32 Mobile Clients.

Figures 4.18 to 4.20 show a similar pattern in that for a fixed number of clients, varying the number of servers does not make a significant impact in the average download time, with exception for TDLS with high density of clients. Since the number of messages is mostly constant when the number of servers increases, the capacity to serve concurrently the download requests is only noticeable when we have 32 clients with TDLS as it avoids the routing node and communicates directly device to device (Figure 4.20). This suggests that the routing device is (also) a bottleneck.

4.5 Conclusion

These experiments allowed us to compare performance and scalability of Wi-Fi, Wi-Fi Direct and TDLS, using the same scenario for all experiments, changing only the technology used.

The results we provided us with very useful information. For instance, it is noticeable that the best solution in terms of performance should be TDLS (in an ideal scenario where every connection uses TDLS), and that both the communication medium and AP/GO have an impact on the average download speed.

The simulation results tell us that increasing the number of servers with a TDLS network is the optimal solution for our problem, and that this works with both Wi-Fi and Wi-Fi Direct.

Chapter 5

Edge Networks to Reduce AP/Server Load

The main goal of this thesis is the study different overlays, focusing on generic types, envisioning the understanding and the comparision of network technologies and their performance behavior on a stadium scenario where content is generated automatically on a remote server connected to the network by an access point (AP) with users fetching files at random times. In this chapter, we investigate whether file distribution among mobile nodes in a local area network can affect the communication performance of the network as a whole.

5.1 Overview of the Experiment

Nowadays, sports teams from all kinds of sports, such as Soccer, American Football or Basketball are installing or expanding Wi-Fi coverage in their venues. Their objective is to meet their supporters expectations and increase their satisfaction by providing a very wanted commodity and providing new ways to experience the game.

More and more, companies such as YinzCam have been installing services that allow die hard fans to enjoy even more their game by generating replays and other premium content to be made available to everyone inside a sports precinct. Although this content is usually quite small, ranging from a few kilobytes (KB) to approximately 5 Megabytes (MB), each section of a stadium has a limited number of APs to which mobile devices can onnect to access such content. For example, the Huwaei made a study to explain how the distribution of their wireless service would be done on the "Estádio da Luz", a 65,000 seats soccer stadium in Portugal, and the results show that only a percentage defined *a priori* of the supporters in each section will have access to the network, and if they all access at the same time, a low QoS is to be expected [27]. The next picture shows in a graphical way how it should work.

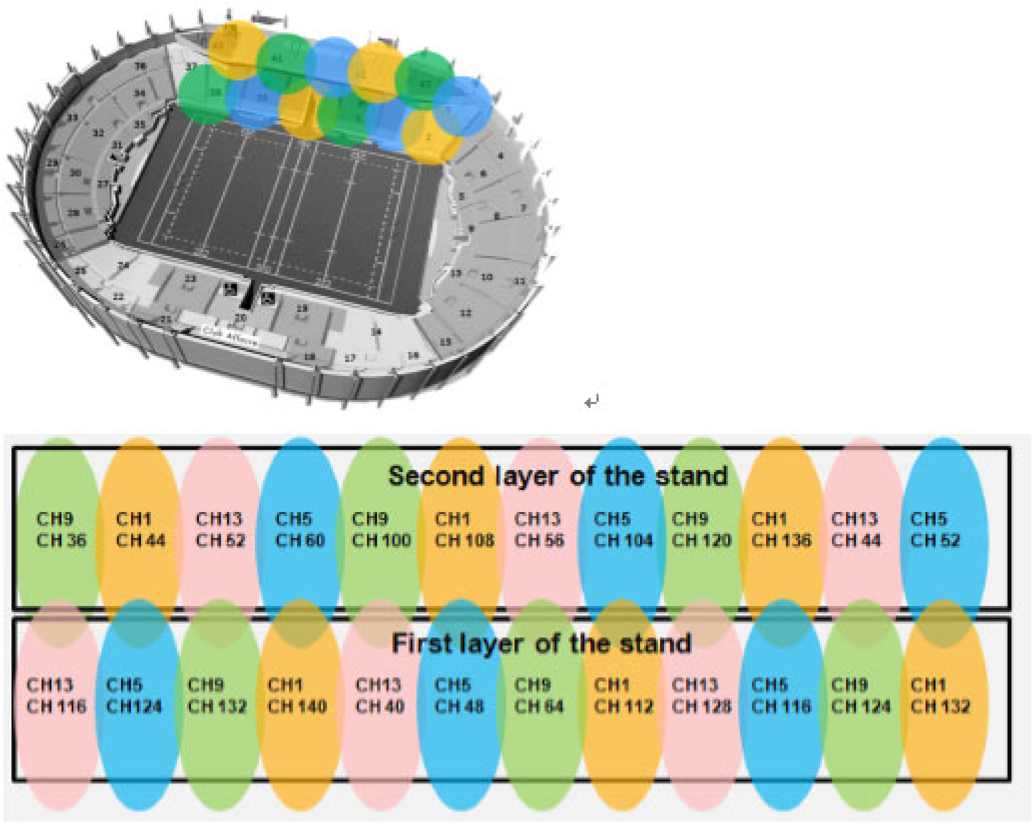


Figure 5.1: Sample Placement of AP per section in Huwaei Agile Stadium Solution, obtained from Huwaei Agile Solution Brochure.

Using the technologies available nowadays on smartphones or tablets, our first goal is to research for strategies that could reduce the number of times each device (or node) needs to access the server (and therefore use the AP as an entry point). In doing so, we expect to increase the capacity of the local network.

This experience scenario was inspired by real-world cases observe in stadiums, where users can use their smartphones to access content provided by an infrastructure in their section. In this scenario the content is generated automatically over the duration of a game, and users have access to all of it, at any time.

To accomplish this study and to be able to observe a reduction in the number of accesses done by the users to the server, we tested essentially the locality of files, without concerning for their size or format.

To reduce the work on the server, we want to create an overlay that is resilient to churn, which is a well known problem in MANETs, while at the same time trying to balance the work done by the local network, to avoid stressing too much some specific nodes given that energy is a strong limitation on mobile devices.

There are plenty of overlay networks that work as MANETs, such is the case of CHORD, CAN or Pastry, but for this particular scenario, none of those networks would be a good choice since we are studying a very specific situation in which churn is an important factor to account for, and it is hard to keep a DHT network updated when nodes keep entering and leaving the network. Another problem with the traditional overlays on the mobile devices scenario, is the fact that technologies such as Bluetooth or Wi-Fi Direct have some limitations and characteristics that would make these kind of networks very unstable and limited.

Pure peer-to-peer networks, or group based networks such as scatternet seem to be good candidates considering the limitations imposed by the technologies used and the problem that we want to solve.

During each experiment, many factors will be taken into account. The main factors being studied in this work are the following:

- Reduction of server accesses over time, varying the number of files and nodes;
- Average number of hops needed to fetch a file;
- Average delay on finding a file;
- Network stabilization time;
- Churn resilience;
- Number of file requests made.

With the simulations we expect highlight the pros and cons for each approach, and be able to indicate which approach is better for the project.

5.2 Technology Limitations & Specifications

In the context of the Hyrax project, there are some limitations on the technology and the level of modifications that can be made either to the operating system, the network stack or the application level.

Android smartphones allow developers to use their Wi-Fi (including Wi-Fi Direct and TDLS), Bluetooth and NFS besides LTE as a means for wireless communication. Wi-Fi modes such as Ad-Hoc mode are not usually available for usage without rooting the devices. Modifications to routing protocols, or network protocols are also not possible without being root.

Since for the first case scenario of the Hyrax project, the goal is to develop a framework that works out-of-the-box without any need for modifications for the OS, in the next simulations we only take into account Wi-Fi and Bluetooth.

In the previous chapter we have introduced the Wi-Fi specification, thus next we describe the Bluetooth.

5.2.1 Bluetooth Specifications

Bluetooth was first released in 1994 by Ericsson with the intent of being an alternative to the *Personal Area Networks (PAN)* using wireless communications over radio frequencies between 2.4 and 2.485GHz. Since its first version, Bluetooth went through several upgrades increasing data rates, introducing compatibility layers with 802.11 and new standards for low energy communication.

In Bluetooth version 3, the term High Speed (HS) was introduced to classify the Protocol Adaptation Layer (PAL) introduced to the protocol stack, which allowed the use of 802.11 protocol to communicate, thus increasing communication speeds. The fourth version of Bluetooth introduced a new operation mode in which the energy consumption is very low. This new mode was called Low Energy (LE).

The following table 5.1 shows the main characteristics of each version (only major upgrades are represented).

As it is possible to observe, there are common limitations and specifications shared across all versions. The most noticeable are the connection limit, the range and link speed, and as such these will be implemented on the simulations.

Version	1.x	2.x	3.0	4.0	4.1	4.2
Link Speed (Mbit/s)	1	3	3 / 24 HS	3 / 24 HS / 1 LE	3 / 24 HS / 1 1 LE	3 / 24 HS / 1 LE
Max. Connections (Slaves)	7	7	7	7	7	7
Range (m)	~ 10	~ 10	~ 10	~ 10	~ 10	~ 10
HS Support	No	No	Yes	Yes	Yes	Yes
LE Support	No	No	No	Yes	Yes	Yes

Table 5.1: Specification Comparison between Bluetooth Versions

5.3 Simulator Limitations & Implementation

Using NS-3 for the simulation has several advantages: well established and verified 802.11 (Wi-Fi) models, mobility and the possibility to expand the platform, but like most current simulation tools it lacks the implementation of the Bluetooth (IEEE 802.15) specification.

Implementing a complete protocol stack is a time consuming and not a trivial task, therefore we decided to emulate the Bluetooth protocol using protocols existent in NS-3. We used a regular Wi-Fi interface in ad-hoc mode and implemented the most relevant Bluetooth limitations, such as connection limit, role and radio distance. The Wi-Fi communication were set at 2.4GHz since that is the frequency Bluetooth uses in its communications.

To perform this implementation in NS-3, each Bluetooth node has a Wi-Fi Ad-Hoc interface using the 802.11g specification. To limit the communication speed, we limited the throughput to 2Mbps, which is approximately the average speed for Bluetooth as specified in Table 5.1. For the communication range, each node follows a mobility model that is used to calculate at each iteration the position of each node. Using this data, we calculate the distance between any two nodes, thus limiting the range to about 50 ft.

Finally, it was necessary to control the number of connections in each node and its role. This is one of the most important limitations of Bluetooth, and to implement it in NS-3 we assigned a variable *role* and a *connection_list* list to each node. The *role* controls the role of a node (*master* or *slave*) and we guarantee that a *master* can never connect to another *master* or a *slave* to a *slave*. The *connection_list* records every connected device to a node. Using this list, we assert that there can never be more than 7 entries, which means that only 7 devices can connect to a single node simultaneously.

This model has been throughly tested, in order to guarantee that all the limitations implemented were working as they should, and the results simulated the real protocol. To do this validation, we used values obtained from real tests performed within the project.

5.4 Experiment Setup and Evaluation Criteria

To obtain data related to each setup that is useful for our study of each overlay, we established a fixed setup that is used in all experiments performed, and measured a set of parameters that provide relevant information for the project.

The experience setup is composed by the following set of conditions:

- Each simulation will run for 6000 seconds;
- The network size will vary in the following exponential manner:
  - 1, 2, 4, 8, 16, 32, 64 and 128 nodes.
- The overlay formation will happen within the first 600 seconds;
- Each node will request a random file available:
  - Upon joining the network;
  - In a random time between 10 and 60 seconds after downloading successfully a file.
- The server:
  - Is available from the first second of simulation;
  - Starts with 0 files available;
  - Adds 5 files every 60 seconds;
  - Will broadcast the file catalog to every node in the network upon updating it.

The evaluation criteria selected to evaluate each overlay are the following:

- Percentage of files obtained locally without resorting to the server;
- Average number of hops needed to find a file in the neighborhood;
- Total number of downloads performed for each network size.

Using this criteria we can then analyze the scalability of each network, the amount of communication needed to find a file in the neighborhood and the reduction on the AP/Server load.

This setup is used in every overlay described below, with exception for the number of hops in the Overlay 2 - Star where this number is fixed (1 hop).

5.5 Overlays

In order to perform the study stated before, we developed four experience scenarios that allowed us to gather important knowledge on the different approaches that can be taken. Each scenario implements a distinct kind of overlay.

The first two overlay scenarios implemented are pretty simple and were done with the intent of being an introduction to the *NS-3*, while at the same time having interest for the project as a comparison with more complex networks. These two models are not very realistic or efficient in terms of energy and file lookup, but could easily be implemented in a real world scenario using Android smartphones and their available communication interfaces.

The last two scenarios try to represent a structured and an unstructured approach to the overlay, having in mind, that using Bluetooth as a communication medium imposes many restrictions on how the network can be formed. In the first scenario (Scenario 3), the network formed is a pure peer-to-peer, and in the second (Scenario 4) a scatternet.

5.5.1 Overlay 1 - Ring Overlay

The Ring Overlay scenario implements a simple ring overlay, where each node in the network is only connected to two neighbor nodes and can only communicate with them and with the server. In this network overlay, in order for a node to find a file it must first search the neighborhood and only after, if the file does not yet exist, will ask the server for it.

This scenario is composed by one fixed server connected to an AP by a Gigabyte Ethernet connection and N nodes connected to each other via Bluetooth and connected with the server using the Wi-Fi physical layer 802.11g. The following picture illustrates this scenario and how each component is connected to the other components.

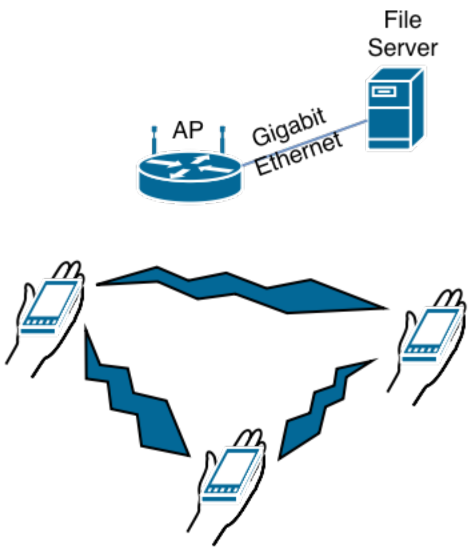


Figure 5.2: Ring Overlay Scenario Architecture

Formation

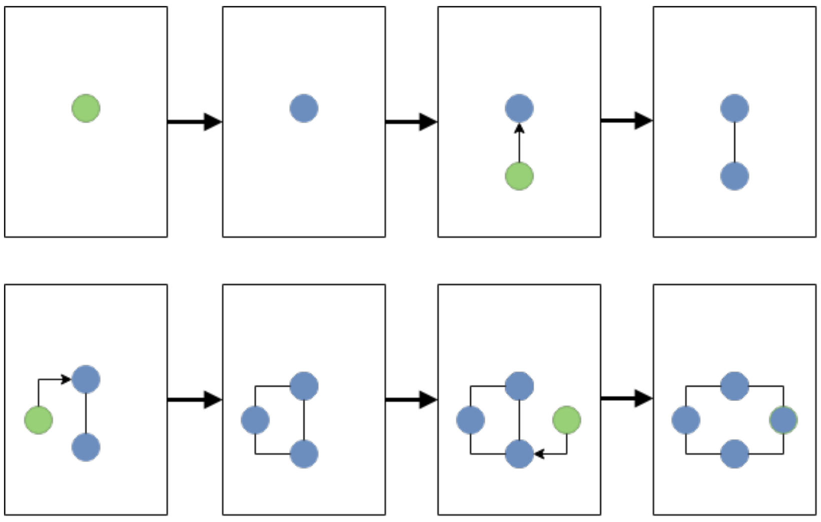


Figure 5.3: Ring Formation Diagram

When a node α wants to join the overlay network, the first step is to search for nearby nodes. By default, every node in range is discovered, although, this could be optimized using the Bluetooth and Wi-Fi Direct service announcement and discovery features, which would enable to discover only available nodes.

After the discovery phase, if there are no other nodes forming an overlay in range, the node α forms a new network without any connections, becoming at the same time available to accept new connection requests, and setting its left and right neighbors to Null. On the other hand, if there are Nodes in range, node α sends a `JOIN` message to a random node β selected from the discovered list.

When a node β already accepting connections receives a `JOIN` request from the node α, two things can happen. node β can decide whether to accept or refuse the connection from node α. Normally, the node β will send an `ACCEPT` message accepting the new node α to the network. If this is the case, then, if the node β has already a *right_neighbor* sends its *right_neighbor* IP to node α in a `CHANGE_LEFT` message, and updates this value to the node α IP and sends the node α IP to the its old *right_neighbor* in a `CHANGE_LEFT` message, else it sends its own IP to node α and sets its *left_neighbor* also to node α since having either the *right_neighbor* or *left_neighbor* as **Null** means it does not have any neighbors yet. In turn, node α receiving an `ACCEPT` message, updates its *left_neighbor* to the node β IP, creating this way the virtual link between both nodes and closing the ring.

When any node already connected in the ring overlay receives either a `CHANGE_LEFT` or `CHANGE_RIGHT` message from one of its neighbors, it updates either the *left_neighbor* or *right_neighbor* to the IP provided in the message. This way, every time a new node joins or leaves the network, it is possible to update the connections on every all needed, in order to keep the ring well formed, and closed.

For the formation of the network, the following Algorithm 1 was used:

Algorithm 1 Ring Overlay Formation

```
function JOIN                                          ▷ Initializes the network join process
    $ip \leftarrow$ DISCOVER                             ▷ Discover a random node in range
    if $ip \neq \emptyset$ then                          ▷ If there are nodes in range
        SEND(‘JOIN’, $ip$)                               ▷ Send a ‘JOIN’ message to $ip$
    else                                               ▷ Else initialize the overlay
        $right\_neighbor \leftarrow \emptyset$           ▷ Setting $right\_neighbor \& left\_neighbor$ as $\emptyset$
        $left\_neighbor \leftarrow \emptyset$
    end if
end function
function LISTEN($request$, $ip$)                       ▷ Listens for incoming $request$ from $ip$
    if $request$ = ‘ACCEPT then                        ▷ If received message is ‘ACCEPT’
        $left\_neighbor \leftarrow ip$                   ▷ Set $left\_neighbor$ as the message sender $ip$
    else if $request$ = ‘JOIN’ then                    ▷ If received message is ‘JOIN’
        $right\_neighbor \leftarrow ip$                  ▷ Set $right\_neighbor$ as message sender $ip$
        SEND(’ACCEPT’, $ip$)                             ▷ Reply ‘ACCEPT’ to $ip$
        if $left\_neighbor \neq \emptyset$ then          ▷ If has neighbors
            SEND(‘CHANGE_RIGHT’+$left\_neighbor$, $ip$)
            SEND(‘CHANGE_LEFT’+$ip$, $left\_neighbor$)
        else
            SEND(‘CHANGE_LEFT’+$own\_ip$, $ip$)          ▷ Send $ip$ own IP
        end if
    else if $request$ = ‘CHANGE_RIGHT’ then            ▷ Change $right\_neighbor$
        $right\_neighbor \leftarrow new\_ip$             ▷ Set $right\_neighbor$ as $new\_ip$
    else if $request$ = ‘CHANGE_LEFT’ then             ▷ Change $left\_neighbor$
        $left\_neighbor \leftarrow new\_ip$              ▷ Set $left\_neighbor$ as $new\_ip$
    end if
end function
```

Churn Resilience and Fault Tolerance

In order to keep the overlay resilient to *churn* and failure, there are some cautions that need to be taken.

If a node α wants to leave the network in a clean way, it must send a `LEAVE` message to both left and right neighbors if they exist, else just leaves the network and stops accepting connections. After sending the `LEAVE` message, node α has to inform its *left_neighbor* of its *right_neighbor*, and vice-versa.

On receiving the `LEAVE` and new neighbor messages, node β updates its local neighbor list, and starts communicating with the new neighbors.

On the other hand, to have some fault tolerance each node maintains a two hop neighbor list (the neighbor of the neighbor), and if the connection to the neighbor fails, it is possible to update the neighbor list by replacing the faulty node by its neighbor.

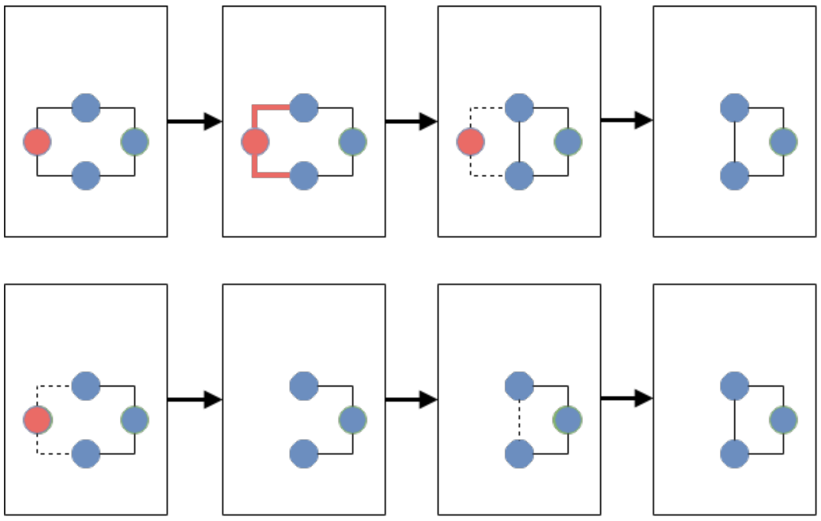


Figure 5.4: Ring Churn Diagram

File transfers and File List Updates

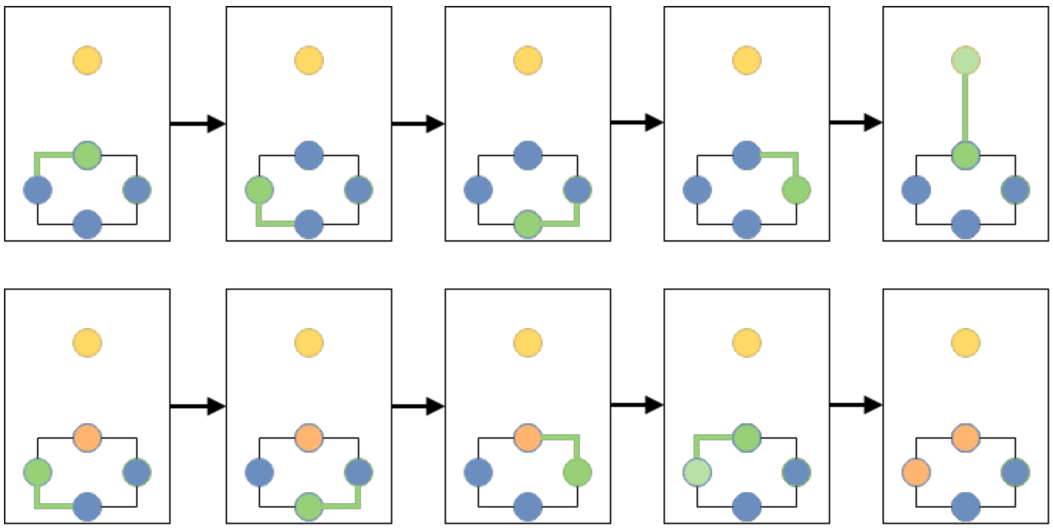


Figure 5.5: Ring Download Diagram

In this ring topology, the file transfer works as a direct connection (we assume every node in the topology can reach any other node as well as the server).

To search files in the local network, a node α needs sends a Token to its *left_neighbor* (node β), containing information about the file being requested, to whom it must be sent and an Unique Identifier (UID). Upon receiving a token, node β registers its UID, verifying if it is a new token, and if it is, checks if it has the requested file. If node β

does have the file, it then sends a SEND message to node α and the file. On the other hand, if node β does not have the file, it forwards this same token to its *left_neighbor* like node α did. This process is repeated until either a node sends node α the file, or the token completes a full circle across the ring, returning to node α, in which case this node will send a GET_FILE message containing the desired file id, to the Server. After the file download is complete, node α is ready to serve the local network with this file in case any other node requests it. This behavior is represented in Algorithm 2.

Besides searching for files in the local network or requesting them to the server, every node needs to know which files are available at any moment. In this Ring overlay, there are two possible approaches to keep the file catalog updated. The first approach is to send a broadcast from the Server every time the file catalog gets updated, informing all the nodes in the network. Another approach implemented was for the nodes to request the latest file catalog from the server, at specific times. These requests can be done at random or fixed times, or every time a node wants to search for a file. At each update, the only information sent from the Server is a *diff* message, containing the new files available since the last update from a node.

Algorithm 2 Ring Overlay File Lookup

```
function LOOKUP($file$)                                    ▷ Starts the file lookup for a $file$
    if $left\_neighbor \neq \emptyset$ then
        $token \leftarrow$ GENTOKEN($file$)                 ▷ Generate a Token for $file$
        SEND($token$, $left\_neighbor$)                      ▷ Send Token to $left\_neighbor$
    else
        SEND('GET_FILE'+$file$, $Server$)                   ▷ Request $file$ from Server
    end if
end function
function LISTEN($request$, $ip$)                   ▷ Listens for incoming $request$ from $ip$
    ...
    if $request = token$ then
        if $token.ip \neq own\_ip$ then                          ▷ Check if Token is not mine
            if ISNEW($token.uid$) then                             ▷ Check if Token is new
                REGISTERUID($token.uid$)                           ▷ Register Token UID
                if HAVE($token.file$) then                  ▷ If this Node had desired file
                    SEND($file$, $token.ip$)                ▷ Send file to requesting node
                else
                    SEND($token$, $left\_neighbor$)                       ▷ Redirect Token
                end if
            end if
        else
            SEND('GET_FILE'+$file$, $Server$)               ▷ Request $file$ from Server
        end if
    end if
end function
```

Results and Discussion

As described before, we obtained three different graphs representing the criteria established for this experiment. In this approach, we obtained values for 3 different simulation scenarios: with no churn, low churn effect and high churn effect.

It is important to define what is “high” or “low” churn. In this simulation, using “high” churn means that each node, upon joining the network will necessarily leave the network. The time each node leaves the network is selected randomly and ranges from 1 second to the last simulation second. Once the node leaves the network, it will rejoin it after a random time that ranges between 1 and $simulation\_time - 5$ seconds. This means that there is a constant churn effect in the network.

For the “low” churn effect, each node in that joins the network, selects a leave time using a random variable. This leave time can range from 1 second to $simulation\_time * 2$ which means, that most of the nodes wont leave the network. Once again, when a node leaves the network, it will use the same formula to calculate the rejoin time. Using “low” churn means that most nodes wont leave the network at any time, and the ones that do leave it, rarely rejoin it later in the simulation.

The graphic in Figure 5.6 shows the results obtained on the percentage of local downloads done in the Ring Overlay, varying the number of nodes exponentially from one to one hundred and twenty eight.

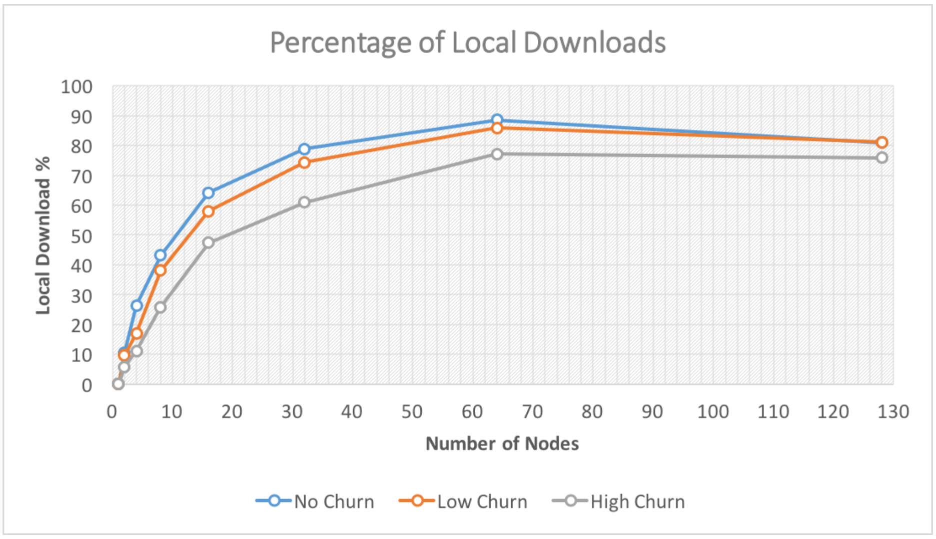


Figure 5.6: Percentage of local downloads in the Ring overlay

The results in Figure 5.6 show that when the network grows to 64 nodes it is possible, using the ring overlay, to download most of the files (between approximately 80% and 90%) from the local neighborhood. It is also noticeable that this value increases from 1 to 64 nodes, after which we verify a decrease. In order to explain this behavior, we need to take in account the number of messages in the network at each time, and the number of hops needed to find a file locally or from the server. The fact that the decrease is lower when there is more churn present, can hint to the fact that the network size is an important factor for these values, since we know that having "high" churn implies that some nodes are always disconnected, thus the network size is smaller.

The next graph in Figure 5.7 shows the results obtained on the average number of hops for the same experiments analyzed before, thus helping us explain these values for 128 nodes.

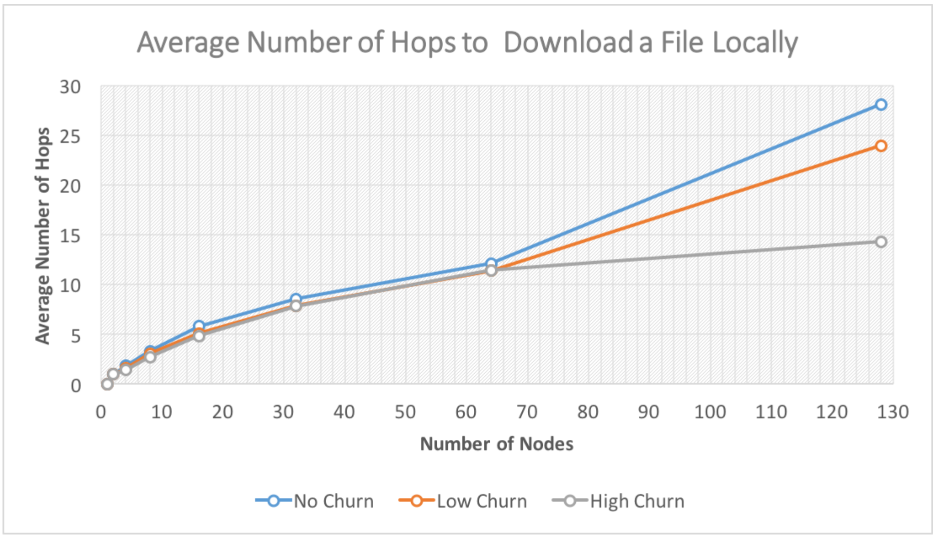


Figure 5.7: Average number of hops in the Ring Overlay

Once again, we can see a linear increase from 1 to 64 nodes and similar average number of hops for each network size for all different scenarios. From 64 to 128 nodes, in both low and no churn scenarios, we verify an abnormal increase on these values.

Comparing these results to the ones on Figure 5.6, we can see that the fact that there are less files available on the neighborhood, which implies that each local download will take more hops to be completed and that there are more downloads directly from the server.

Finally, Figure 5.8 shows the results obtained on the total number of downloads performed on each scenario.

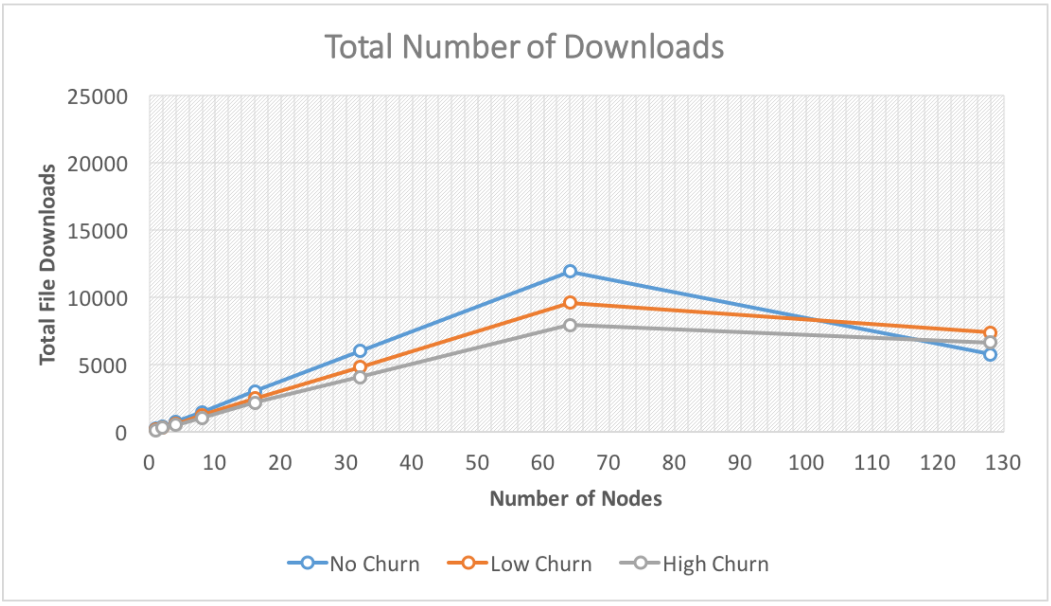


Figure 5.8: Total number of downloads in the Ring overlay

In this graph, it is possible to see that up to 64 nodes, the total number of downloads performed increases linearly which corroborates with the results obtained before for these same scenarios and network sizes.

From 64 to 128 nodes we verify a decrease on this number on all scenarios. This decrease is much more accentuated when there is no churn, and it is less noticeable when there is an high amount of churn.

To explain these results, we need to remember the fact that in order to download a file, each node sends a token that has to travel through all the nodes or until is found locally. This means that for a network of 128 nodes, there is a very high number of communications being done at each moment, thus saturating the medium and reducing a lot the available bandwidth. This reduction of the bandwidth means that for each hop, the message will take much more time (T) than for the smaller networks, and since the network has now 128 nodes, it will have to do 128 hops (H) before going to the server.

Having a network of size S, we can define the time a download takes using the following expression:

$$DownloadTime = \begin{cases} T * H,\ C, H \geq 1 & \text{if file is local} \\ T * S + 1,\ T, S \geq 1 & \text{if file is not local} \end{cases}$$

From this equation we can relate both network size and hop time to the total download time, and therefore to the decrease of the number of downloads if T increases a lot, comparatively to the simulation time, which only occurs after 64 nodes. For instance, with 128 nodes, each download from the server can take more than the simulation time, thus being canceled when the simulation finishes.

To explain the large increase of hops, we need to take in account the fact that the simulation runs for 6000 seconds, that only after 600 seconds all nodes are connected and that nodes start downloading content as soon as they enter the network. This means that when the medium is still not saturated, i.e. during the formation period, nodes are able of downloading files, mostly from the server, thus populating the local network. After the formation period, when the medium becomes saturated, each node takes a very long time to decide whether to go to the server or to receive the file from a neighbor. This means that there is a low population of files locally and it is very rare for a node to download from the server, thus most local downloads will take more hops to be accomplished.

The next graph in Figure 5.9 shows the effect. It is possible to see that for 128 nodes, after the 600 seconds (when the overlay is completely formed) there is a decrease of downloads, and almost every download is done locally.

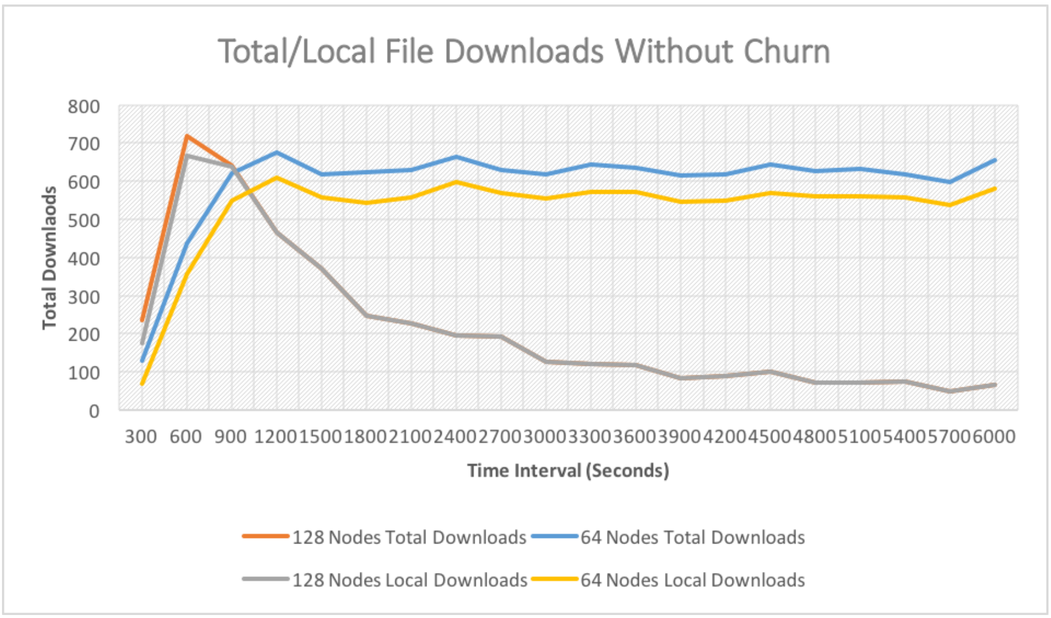


Figure 5.9: Total/Local number of downloads in the Ring overlay

In conclusion, the ring overlay is not scalable for networks larger than 64 nodes, which makes it a not suitable solution for the problem we are trying to address.

5.5.2 Overlay 2 - Star

The star overlay was though as an introduction to a basic hierarchical topology, that could allow us to compare a full decentralized approach to a more organized one. There are no hard bounds on the number of connections that can be made from a node to another, and each star group is independent of every other.

The basic idea behind this overlay, it to have a node α behaving as a Group Owner or Master, and N group members or slaves. The group owner is the only node that can connect to any other node, while the other nodes can only connect to the Group Owner.

In this overlay, there are many combinations of scenarios that can be tested. For instance, the way file lists are updated or how the files are requested, can be done in several different ways.

As with the ring overlay, every node communicates using Wi-Fi 802.11g, and the Server is connected to the AP via a Gigabit Ethernet link.

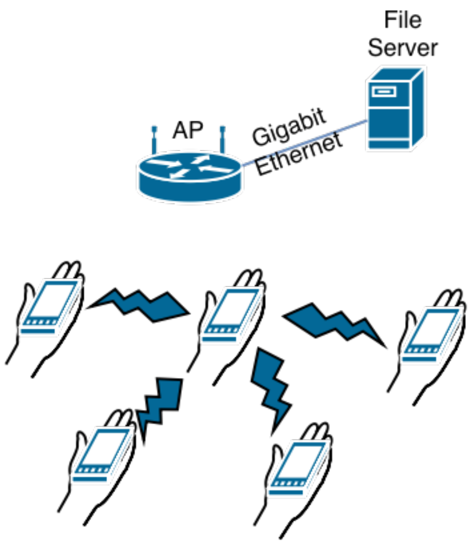


Figure 5.10: Star Overlay Scenario Architecture

Formation

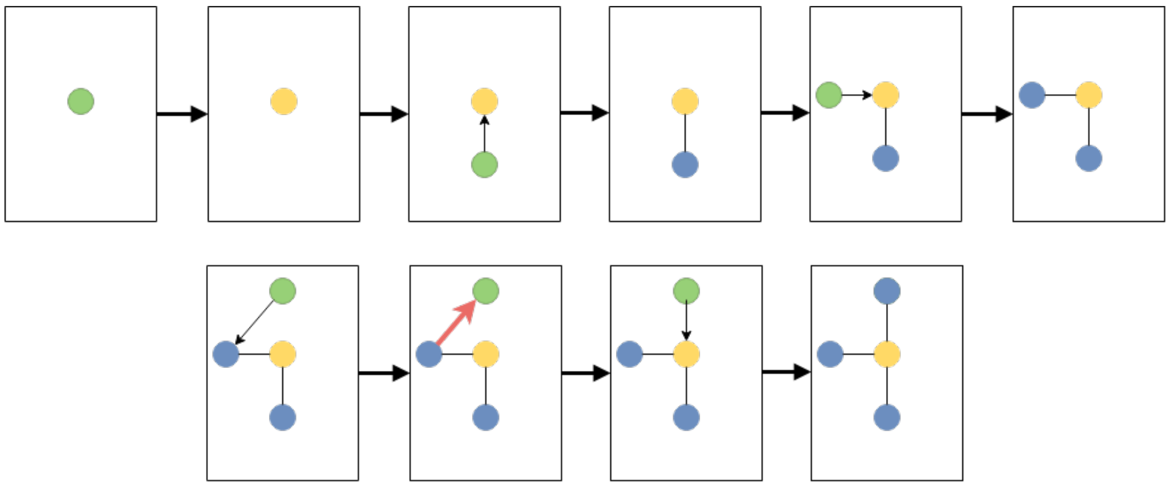


Figure 5.11: Star Formation Diagram

To form a star overlay, one node must behave as a group owner. For simplification reasons, the selection of the group owner node is made automatically assigning that role to the first node to form the overlay.

When a node α wants to join this overlay, the first step is to perform a discovery, finding every node in range. If there are no nodes in range, node α becomes a group owner, and starts accepting connections to its group. On the other hand, if there are nodes in range, a node β is selected randomly, and a `JOIN` message is sent to it. If the node β is a Group Owner, it can either `ACCEPT`, or `REFUSE` the new connection. If the node α receives an `ACCEPT` message, it joins the network, else it retries with other nodes in range until it is accepted by a group, or running out of nodes, in which case, it forms a new group.

When a *Node C*, that is not a Group Owner receives a `JOIN` message from a node α, it replies to this requesting node with the IP of the Group Owner. Since *Node C* can not decide if new connections are accepted being or not by its Group Owner.

Algorithm 3 represents the start formation.

Algorithm 3 Star Overlay Formation

```
function JOIN                                         ▷ Initializes the network join process
    $ip \leftarrow$ DISCOVER                          ▷ Discover a random node in range
    if $ip \neq \emptyset$ then                       ▷ If there are nodes in range
        SEND('JOIN', $ip$)                            ▷ Send a 'JOIN' message to $ip$
    else                                              ▷ Else initialize the overlay
        $connections \leftarrow \emptyset$            ▷ Setting $connections$ as $\emptyset$
        $role \leftarrow master$                      ▷ Setting $role$ as master
    end if
end function
function LISTEN($request$, $ip$)                      ▷ Listens for incoming $request$ from $ip$
    if $request$ = 'ACCEPT then                       ▷ If received message is 'ACCEPT'
        $master \leftarrow ip$                        ▷ Set $master$ as the message sender $ip$
        $role \leftarrow slave$                       ▷ Set $role$ as $slave$
    else if $request$ = 'JOIN' then                   ▷ If received message is 'JOIN'
        if $role == master$ then                      ▷ If is $master$
            SEND('ACCEPT', $ip$)                      ▷ Reply 'ACCEPT' to $ip$
            $connections += ip$                       ▷ Add $ip$ to $connections$
        else
            SEND('MASTER'+$master$, $ip$)             ▷ Send $ip$ $master$ IP
        end if
    else if $request$ = 'MASTER' then                 ▷ Contact $master$
        SEND('JOIN', $request.ip$)                    ▷ Send a 'JOIN' message to $request.ip$
    end if
end function
```

Churn resilience

In this overlay there is a single point of failure that is hard to overcome in an efficient way, and for this reason, we did not implement any scenario in which the group owner could fail or leave the network. As for every other node, leaving the network in a clean way means sending a `LEAVE` message to the Group Owner, which in turn, removes all file entries of the leaving node from its local file location table. If another node fails and does not inform the group owner, its entries will only be remove after another node tries to connect to the failed node, and fails, informing the group owner of that fact.

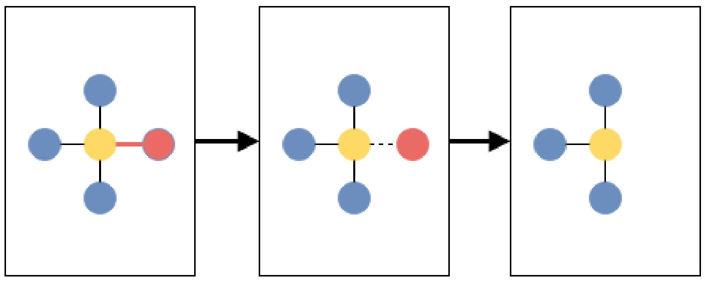


Figure 5.12: Star Churn Resilience Diagram

File transfers and File List updates

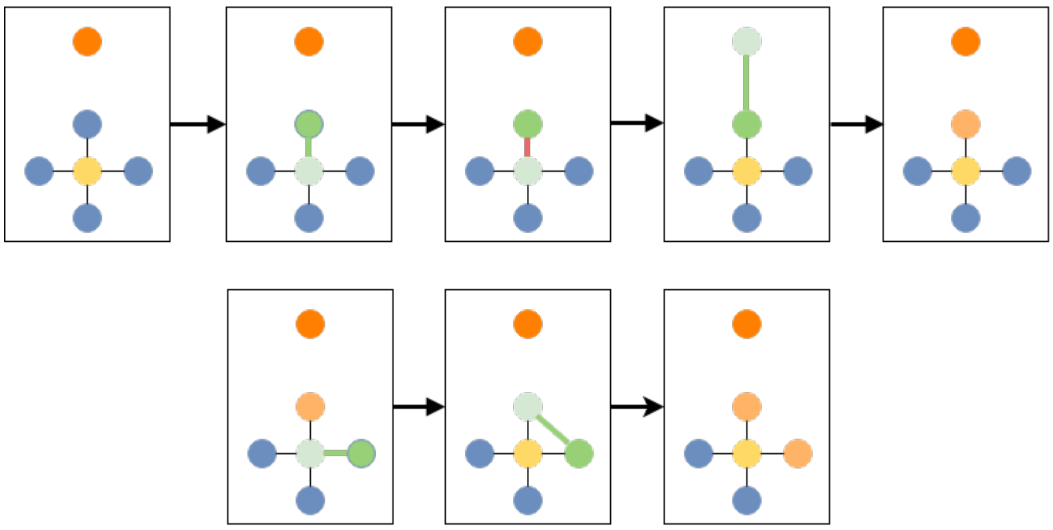


Figure 5.13: Start Formation Diagram

The lookup of files in the Star overlay is much simpler and efficient than the Ring overlay. This happens because the Group Owner node contains a list of every file available on its local group, and when a Group Member node wants a file, it just has to sent a message to the Group Owner.

In order to look for a file, a node α send a `WHO_HAS` message to its Group Owner. This message contains, the request and the identification of the file, being requested. Upon receiving this message, the Group Owner checks its local file list, and verifies whether the files exists in any local node, and if that is the case, it replies with a `REQUEST_TO` containing the IP of a random node that is known to have the requested file. On the other hand, if the file hasn't been fetched yet by any local node, the reply is `GOTO_SERVER`.

When node α receives a `REQUEST_TO` message, it sends a `GET_FILE` message, informing the other node of its desire to obtain the file. If the Group Owner reply was a `GOTO_SERVER`, node α will request the file from the Server.

After getting the file, node α has to inform the Group Owner about it, and to do so, it sends a `I_HAVE` message, containing the file id. Upon receiving this information, the Group Owner updates its local table, adding the node α as a source for the file.

While finding files is fast, keeping up-to-date the available file lists, can be done in several different ways. If the main objective is to offload as much computation from the server (and AP) as possible, instead of broadcasting the file list updates to the whole network, the server sends the updated file list to the Group Owner which in turn takes care of distributing the information across its Group Members. This can be done, by broadcasting the information, or by waiting for each Group Member to request the latest file list.

Results and Discussion

In this approach, we obtained two different graphs representing the percentage of downloads performed from each node neighborhood and the total number of downloads realized for each network. As with the Ring Overlay, we obtained values for 3 different simulation scenarios: without churn, low churn effect and high churn effect.

Figure 5.14 shows the local file discovery results we obtained using the Star overlay.

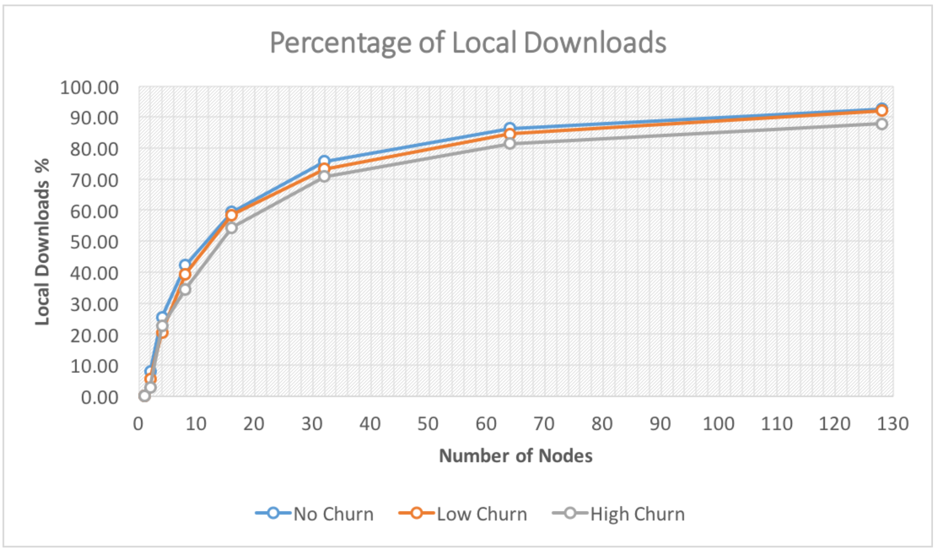


Figure 5.14: Percentage of local downloads in the Star overlay

We can see that the behavior of all curves is the same, reaching approximately 100% when the network has more than 64 nodes.

This happens because each node takes very few time (few ms) to search the local network and if needed resort to the server, which means that for larger networks, almost all the content available will be cached in the neighborhood.

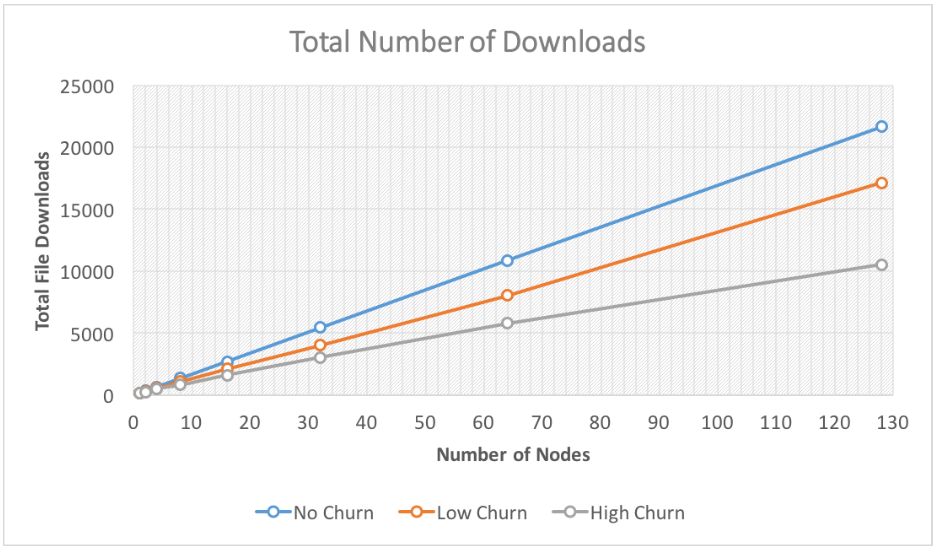


Figure 5.15: Total number of downloads in the Star overlay

In Figure 5.15 we observe a linear growth in the total number of files downloaded, and we see that the churn affects this number.

There is no need to show results regarding the number of hops due to the fact that it is either 1 hop for a local download or 0 for a server download.

In this network we do not see a degradation for the larger network, thus we can conclude that it is scalable. The only problem with this approach is the huge amount of work done by the *master* node, having a single point of failure and the fact that currently this approach can not be implemented on smartphones, due to technology limitations.

5.5.3 Overlay 3 - Peer-to-peer

The peer-to-peer overlay network implemented for this study, tries to mimic a true, unstructured architecture, where each node can connect with another, being limited only by the amount of connections accepted per node. This is a realistic overlay that can be implemented on any Android mobile phone without having to rooting the device. To keep this overlay compliant with the Bluetooth limitations, the limit of accepted connections per node is 5 (Bluetooth allows up to 7) in order to keep 2 open connection slots for any necessary communication outside the network formed.

In this overlay each node is classified either as *master* or a *slave*, as it happens in

Bluetooth. The masters can only communicate with slaves and vice-versa. This does not affect much the network formation, but it can be helpful to test different scenarios of communication.

Being a peer-to-peer network over a MANET, there were precautions that had to be taken and some techniques to transmit data in a controlled manner, such controlling number of hops. In order to communicate, each node can be used to send a message to a specific direct neighbor or perform a controlled flood that propagates a message up-to a maximum depth, that can be defined at runtime. As with other overlays, every node in this network is able to access the Server if necessary.

In the Figure 5.16 it is possible to see the architecture layout of this overlay.

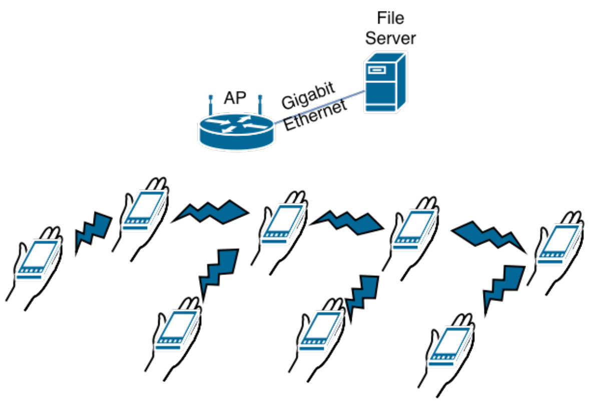


Figure 5.16: Peer-To-Peer Scenario Architecture

Formation

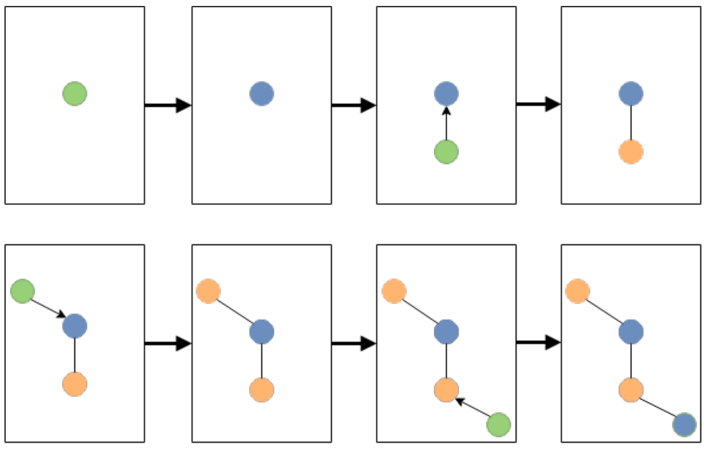


Figure 5.17: Peer-To-Peer Formation Diagram

When a node α wishes to join a network, the first step taken is to perform a discover, in order to find every node available in range. If there are no nodes in range, node α forms its own network, and assigns itself the role of *master*. On the contrary, if there are some nodes in range, node α selects one at random, and sends a `JOIN` message.

Meanwhile, if a node β receives the `JOIN` request, it can accept or refuse the new connection. By principle, if node β has less than 5 connections, it will send an `ACCEPT` message and its current role to node α, and insert the node α address in its connections. At the same time, when node α receives the `ACCEPT` will add node β address to its connections, and assume the contrary role of node β. If on the other hand, node β can not accept any more connections, it will send a `REFUSE` message to node α, which in turn, will ignore this node, and restart the process until connecting or running out of options, case in which it would create a new network.

This random behavior and lack of structure make it a very basic, fast and lightweight protocol to form the network, but as it is possible to verify in the Figure 5.18, the density of the network can be low in some areas, which may lead to an higher amount of Server connections, than on a denser network.

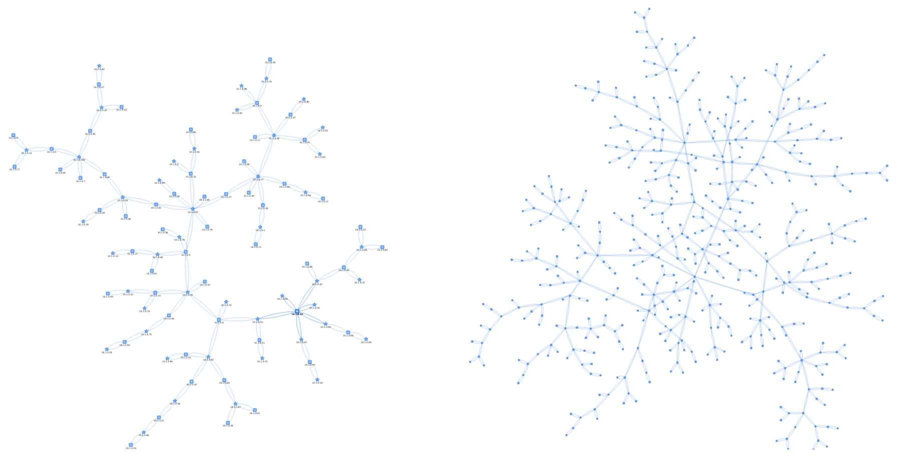


Figure 5.18: Peer-To-Peer Network with 100 and 500 Nodes

Churn resilience

In this network, because of the *master/slave* restrictions, keeping a fault tolerant network is not trivial. The chosen approach to surpass this problem was to keep in each node information about all the neighbors in order to reconnect groups if either a *master* or a *slave* leaves the network. This redundancy of connections means that if any node leaves the network, the remaining nodes are able to reconstruct it.

The next diagram, shows an example of the approach described before.

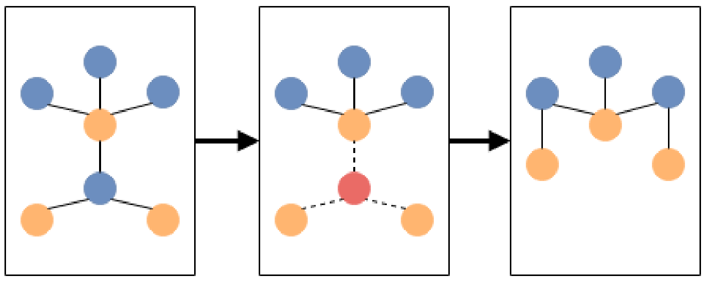


Figure 5.19: Peer-To-Peer Churn Resilience Procedure

File transfers

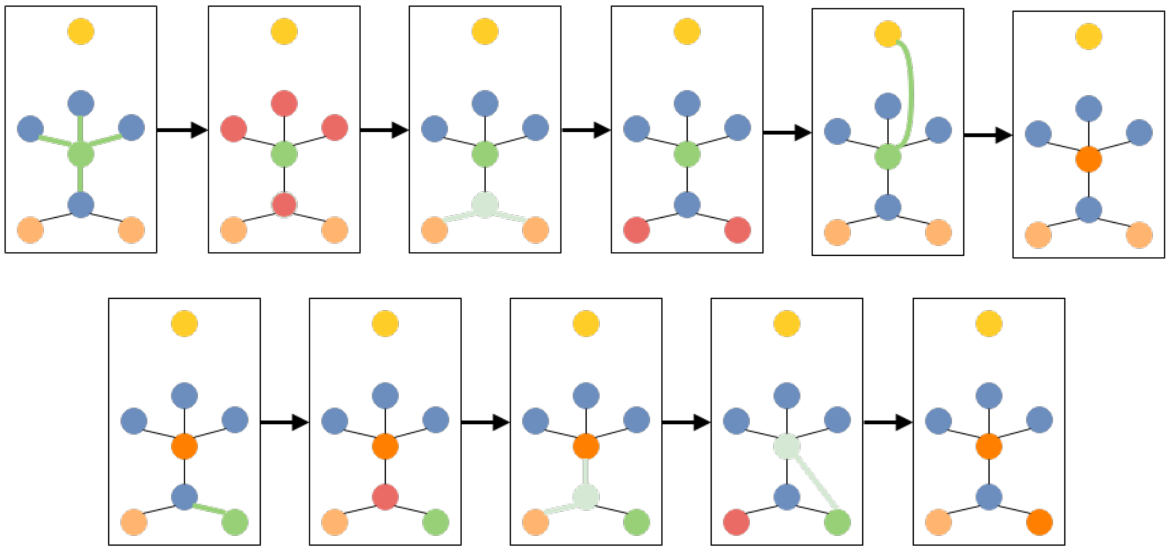


Figure 5.20: Peer-To-Peer Formation Diagram

In order to search for files in the local overlay, two different approaches were taken.

The first approach implemented was to use a controlled flood. When ever a node α wants to find a specific file, it sends a token containing the desired file id, its own IP, a UID, and a hop counter (starting at 0). Once the token is generated it is sent to every connection node α has. Upon receiving a token, any other node β will first check if the token is new (this is to avoid loops) and if it is, will check whether it owns the requested file. If the file exists in a Node, it will be sent to node α directly, and if it does not, node β will increment the hop counter and check if the maximum number of hops has been reached. If the maximum number of hops hasn't been reached yet, the token will be forwarded to every other connection in node β, except to the node that sent the message to it.

Since there is not control on whether the file exists in the local network and the lookup has ended, or if the delays on the network are very high, node α has a timeout after which it will discard the request and attempt to download the file from the server. If after the download has started, either locally or from the Server, node α receives another request for the same file, it will reject the connection, avoiding this way unnecessary downloads. In this approach all the message routing and redirects are made at the application level.

In the second approach implemented, each node keep a list of its neighbors neighbors, thus enabling direct communication in a two hop range. To create and maintain the neighbor neighbors table, each time a node α connects to a node β, it receives node β neighbor list.

Once the node is connected to the network, each time it receives a new connection or loses a neighbor, it must send its updated neighbor list to every neighbor, thus keeping every table updated.

Assuming that each node now contains two lists of nodes, each time a node α wants to search for the file, it will simply send a message to every node contained on both lists, requesting a file. This way, with only 1 hop, node α can reach two levels of depth in the overlay. As before, if a node β has the file node α requested, it will send the file to it. Again, node α does not expect a message from every node, informing that the file is not available, so it will use the same timeout mechanism implemented in the first approach, and get the file from the server if needed. Since it is not easy to synchronize the files available on each of the nodes in the node α lists, if more than one node (or the Server) attempts to send a requested file to node α after it has already started getting it from another source, the connection will be refused.

Similar to the Ring overlay, to keep the file catalog updated in this overlay, either the Server broadcasts the updates to every node in the network, or each node requests the updated catalog from the Server.

Results and Discussion

To study the behavior of the peer-to-peer approach in terms of scalability, quality of service and benefits for the proposed scenario, we obtained results for the following setups: Using a tabled approach, allowing only 1 hops, 2 hops, 3 hops, 4 hops and finally without hop amount limitation. Thus, we have six different setups to analyze. The timeout for a request was set at 0.5 seconds, after which time the request is forwarded to the server.

This first graph contains the results concerning the average percentage of downloads that did not required access to the server.

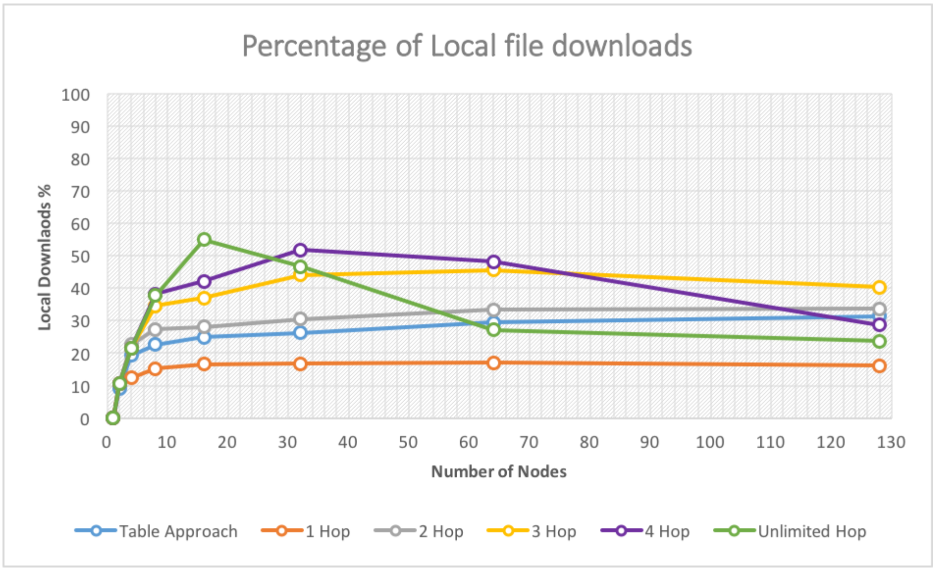


Figure 5.21: Percentage of local downloads in the Peer-To-Peer overlay

It is possible to see that for setups where the number of hops is limited to a low amount, such is the case of the tabled approach or the limited to 1-3 hops, the percentage increases with the smaller network, reaching a plateau that is maintained through the larger network sizes. On the other hand, we can verify that allowing a larger number of hops appears to be beneficial for small networks, but its performance decreases a lot when these network get larger.

This indicates that allowing an high amount of hops to search files can be beneficial on small networks but it is not scalable.

In the following graph, we analyze the average number of hops performed to find files in the neighborhood. It is important to keep in mind that there is a timeout in every file request which may influence the maximum number of hops done if the network becomes saturated and slow.

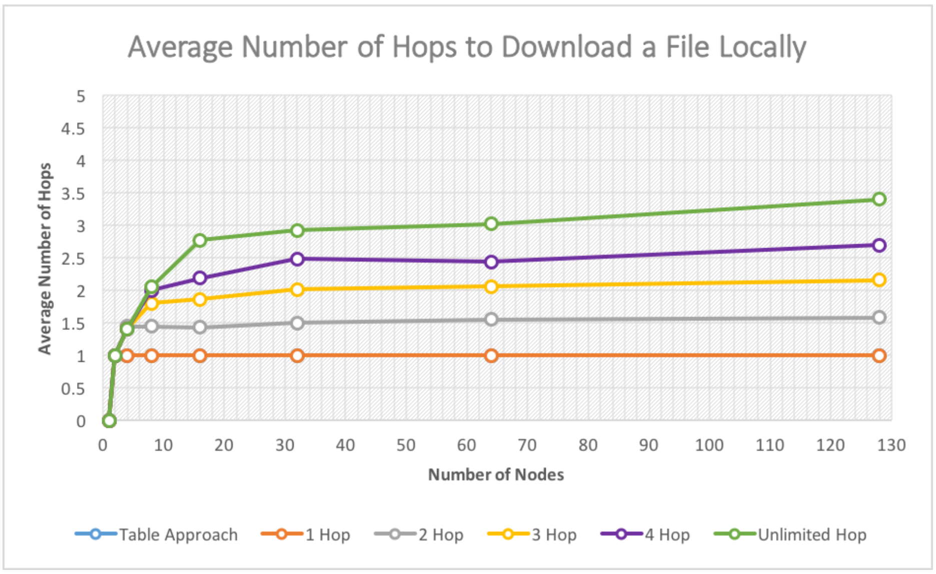


Figure 5.22: Average number of hops in the Peer-To-Peer Overlay

The results displayed in this graph show that the average number of hops increases in a linear fashion from one to four hops. It is also noticeable that this behavior is less constant when we increase the number of hops allowed. Finally, we can see that allowing unlimited hops does not increase in a significant way the depth of search performed by each node, and may have adverse consequences. The following graph allows us to verify the consequences of increasing the number of hops allowed.

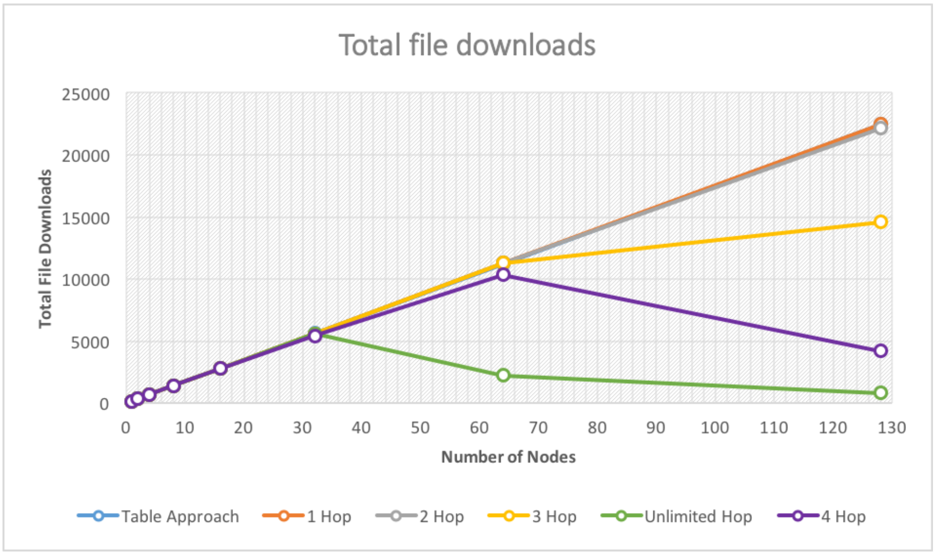


Figure 5.23: Total number of downloads in the Peer-To-Peer overlay

Analyzing Figure 5.23, it is possible to notice a similar behavior to the one in Figure 5.14 for each curve. It is clear that increasing the number of hops each message can travel leads to a flood of the network which reduces its speed, thus reducing the scalability of this overlay.

The results showed before demonstrate that flooding the network is never a good alternative. If we want a scalable network we have to limit the search area to a small value to avoid flooding.

5.5.4 Overlay 4 - Scatternet

The Scatternet overlay implemented in this study, was based on the Bluetooth scatternets, which are networks formed by multiple groups composed by a *master* and several *slave* nodes called piconets, connected between themselves using shared slave devices called bridges. In this implementation, the objective was to simulate the behavior of Bluetooth connections, thus each slave must be connected to n *master* nodes where $7 > n > 0$, and each *master* can only connect to n slave nodes.

This overlay serves as a realistic scenario, that can be used, as happened with the peer-to-peer overlay, with most of smartphones without having to modify the devices, in which there is a structured architecture and lots of testing scenarios.

The Figure 5.24 shows the architecture used in the NS-3 to simulate this network overlay.

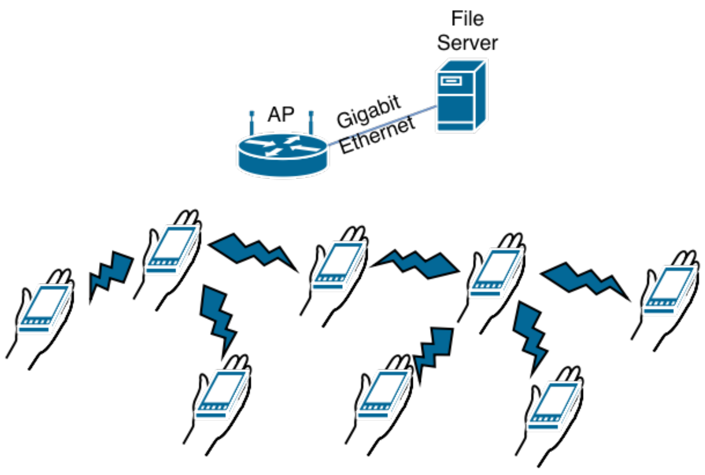


Figure 5.24: Scatternet Scenario Architecture

Formation

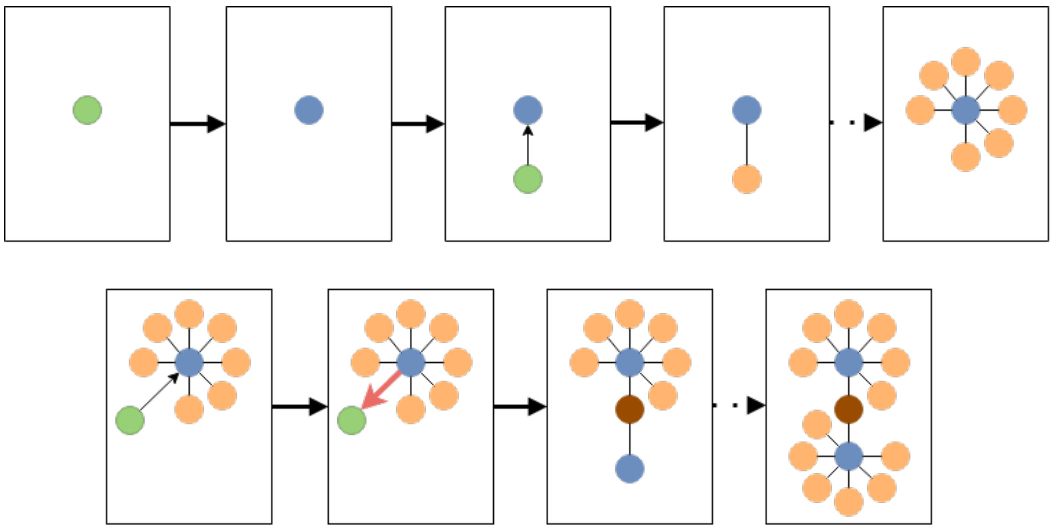


Figure 5.25: Scatternet Formation Diagram

The scatternet formation can be divided in two parts. The first part shows how to form each piconet, and the second shows how to connect different piconets in order to form the larger scatternet.

The piconet formation is somewhat similar to the Star overlay formation, only with more restrictions and roles in each node. The first node to enter the network, becomes a *master node* and starts accepting connections. When another node α wants to enter the network, it will perform a discovery. Then it will send a `JOIN` message to a random selected node in range. This discovery service will find every node available, thus if a *slave* node receives the `JOIN` request, it will reply to its *master* IP. If the selected node is a *master* node and still hasn't reached the maximum amount of connections, it will send an `ACCEPT` message to node α, and will be add it to the piconet. When node α is accepted in the piconet it will set its role as *slave*, and its *master* node will add it to its connection list.

Since the objective of this overlay is to build the most dense possible network, when a node α attempts to connect to another node β, that is a *master node*, and receives in return a `FULL` message, it will have to connect either to a *slave* node, building a *bridge* and forming the scatternet, or just assume the role of *master*, ignoring any possible connection to the existent groups.

To build the scatternet, bridges must be established to connect the groups. In order to control the status of the neighbor groups, each *group master* has a list of neighbors and their current status (**full/not full**).

When a *Group* γ is full, and a node α attempts to join it, the master of *Group* γ will check his list of neighbors to see if any of the current neighbor group are available (**not full**) to receive node α as a member. If there is a group with room left for node α, *Group* γ *master* will send a `GO_JOIN` message, containing the neighbor group master IP address, to this node. If on the other hand, there are no neighbor groups with availability to connect to node α then *Group* γ *master* will choose a random group member with enough connection slots available to form a new bridge and send a `MAKE_GROUP_WITH` message to node α, which upon receiving will set its role to *master* and send a `MAKE_GROUP` message to the assigned bridge node thus creating a new connected group.

Every time a *group master* tells a node to form a new group, it will add this new node IP to the neighbor group list as it became a new *group master*, and sets its status to **not full**. At the same time, the new *group master* will add the neighbor *master* IP to its neighbors list and set its status to **full**.

Finally, when any group becomes full it will send an `IM_FULL` message to all its neighbors, informing them to update their local neighbor list status to **full**.

Although this scatternet formation method cannot ensure that all groups will be full before creating new ones, the node density will be at its top capacity throughout most of the network. Only groups in the extremities have a chance of not being full as can be seen on the networks formed automatically using this method and shown in Figure 5.26.

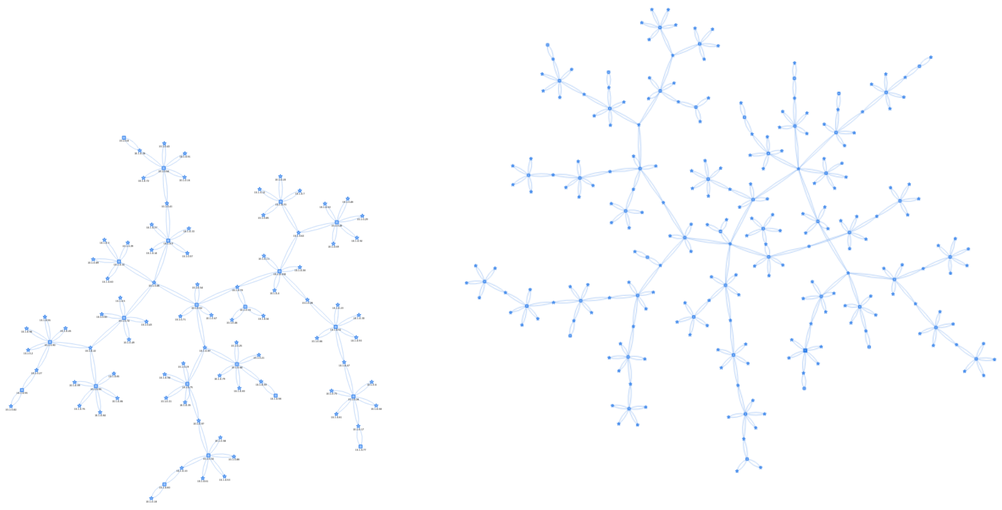


Figure 5.26: Scatternet Network with 100 and 250 Nodes

Churn resilience

To keep this network churn resilient and failure tolerant is quite challenging, as there are multiple single points of failure that have to be prevented. For instance, if either a *master* node or a *bridge* node leaves the network in any way (failure or clean), the connection between groups and nodes is lost.

In order to overcame the problem of a bridge node leaving the network, a similar approach to the one implemented for the peer-to-peer network could be used. This solution implies that each *group master* has knowledge about neighbor group slaves, and when a bridge is lost a new connection is established to one of those slaves. For instance, if a *Master Node* from a *group* α knows the IP of all (or some) of the *slave* nodes from a neighbor *group* β and the *bridge* node that connects groups α and β leaves the network, it is possible to relink both groups by having the *group* α *master* connect to one of the *group* β slaves.

Another solution for this problem is for the *master* node of a group to request a slave node to connect to the new disconnected *group master*, thus simplifying the recovery process. In this approach, it is assumed that each *master* knows all its neighbor group *master* IPs.

When a *master* leaves the network, its slaves have to search for new groups to join to.

File transfers

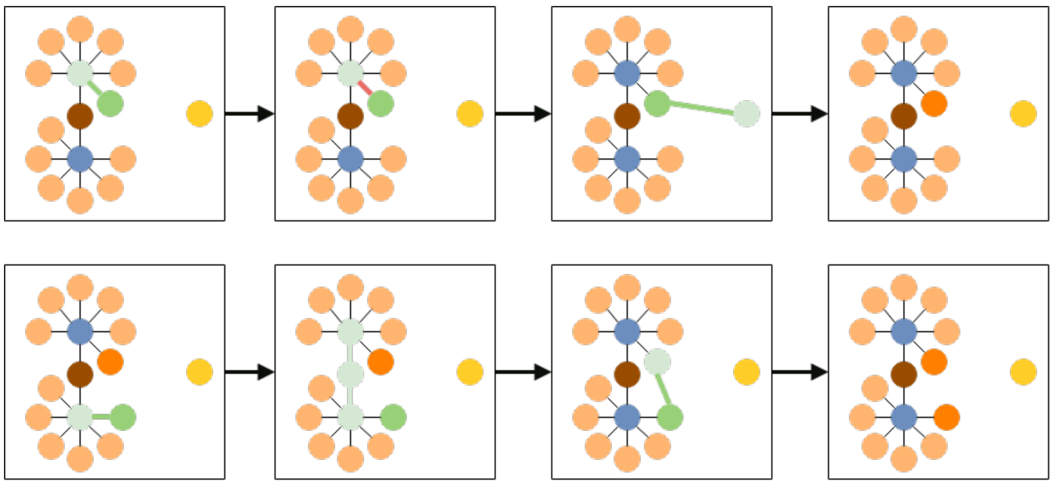


Figure 5.27: Scatternet Download Diagram

In the Scatternet overlay, there are two ways of looking for a file. The first and simpler method, does not require the groups to be connected using the bridge devices as it only performs a search in the local group, while in the second method, the search is expanded to neighboring groups, allowing for a larger range of local content sources.

Similarly the *Overlay 2 - Star*, in each group of the *Overlay 4 - Scatternet* the *master* node is responsible for recording every group member files, which means that it will also be the node responsible for informing nodes about the location of the files they might request.

Although there are two strategies of performing file lookups, the first step is common to both approaches. When a node α wishes to download a *file X* it has two options. If it is a *slave node* it sends a `WHO_HAS` message to the *group master*, otherwise it simply checks is local file location table for *file X*.

The second strategy is where the two methods differentiate. In the first approach, since the search is only local to the node α group, if the file is available and this node is a *slave*, the *master* will reply with a `ASK_FILE_TO` message containing the IP address of a random selected node that has *file X* available. Upon receiving the message, node α will then send a `GET_FILE` to the provided IP, and expect to receive the file in response. If node α is the *master* it will skip the `ASK_FILE_TO` step, and simply send the `GET_FILE` message to a random selected provider of *file X*.

If the file is not available on the local group and node α is the *master*, it will request the file from the *Server*. If on the other hard, node α is a *slave*, the *master* will reply with a `GO_TO_SERVER` message and node α will request the file from the *Server*.

When the download if is finished, node α will either inform the *master* that the file is now available in its node and the *master* will register it in the local file location table, or simply records its own IP in this same table otherwise. After this step, the file transfer is finished in the first approach.

Using the second approach, where the file lookups are not restricted to the local group, upon receiving the `WHO_HAS` message, the *master* node will first search the local file table for the file, and if the file is not available in it, it will send a `DO_YOU_HAVE` message to all its neighbors containing information about the requested file and the IP of the node α, and set a timeout for a response from one of them.

When a *master* node receives a `DO_YOU_HAVE` from a neighboring group *master*, it will check the local table, and if it has the file, it will send it directly to node α. If the *master* does not have the file, but one of its group members has it, it will send a `HOP_ASK_FILE_TO` message to node α containing the IP node containing *file X*. On both situations, this *master* node will send a `I_HAVE_SENT` message to the neighbor *master* that originally requested the file, in order to inform it that node α can obtain the file locally.

Upon receiving a `HOP_ASK_FILE_TO`, node α will send a `HOP_GET_FILE` to the IP contained in that message and wait for the file in response. The routing of the message across the master and the bridge until reaching the destination is done automatically using a MANET routing algorithm.

If the node α *group master* reaches the timeout set on the first step without receiving a `I_HAVE_SENT` message corresponding to the request done, it will inform node α to download the file from the *Server*.

Once again, upon finishing the file X download, node α has to inform its *master* node thus finishing the file transfer process.

Results and Discussion

For the scatternet overlay three different results were obtained and are shown in the following graphs 5.28, 5.29 and 5.30.

In the first graph, the results shown are related to the percentage of files that are obtained locally without accessing the server for different network sizes.

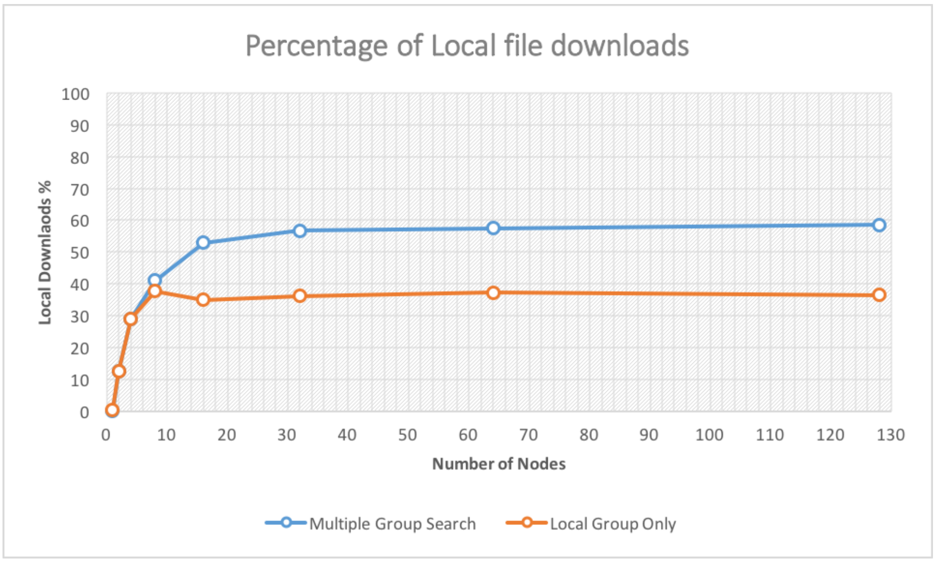


Figure 5.28: Percentage of local downloads in the Scatternet overlay

From the analysis of these results, it is possible to verify a tendency of reaching a plateau when the network increases. This is more noticeable for both cases when the number of nodes reaches 16, 32 and upwards. It is also clear that if the main purpose of a solution is to reduce the amount of times the devices have to access the server, enabling the discovery on neighbor groups is beneficial as the plateau is higher, and in this case, higher is better.

But increasing the area of search of local files has an impact on the number of messages exchanged in the network. Although the next graph does not directly refer to the number of messages exchanged, by analyzing the number of hops needed to find a file in the local network, it is possible to deduce the message amount.

It is important to notice, that the number of hops represented in the following graph, is only calculated when the file is reached locally. If the file is downloaded from the server, the number of hops is not calculated, and we assume the worst case.

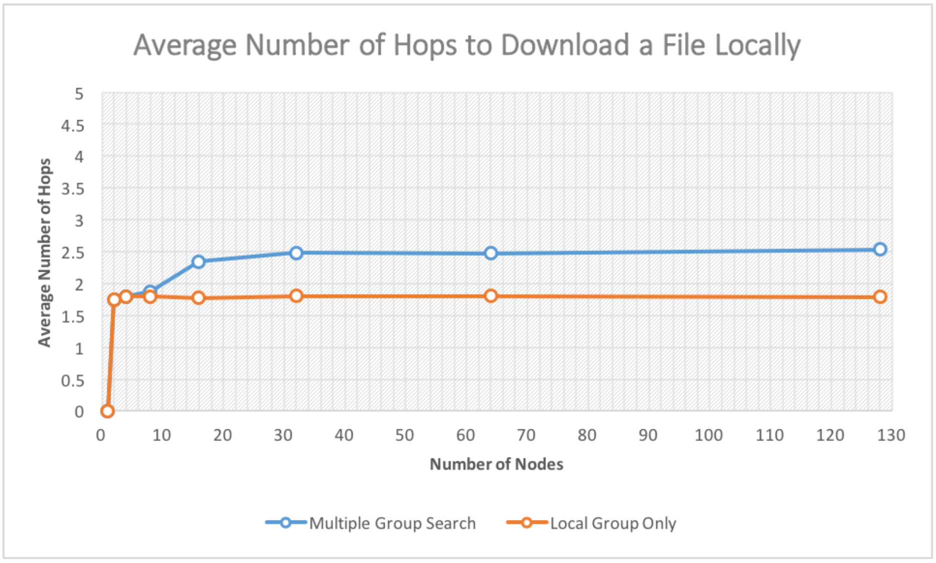


Figure 5.29: Average number of hops in the Scatternet Overlay

Once again, it is noticeable the fact that both approaches show the same behavior for small networks, but when the network size increases, the local search shows better results compared to the local neighborhood search. In this graph, the smaller the value the better the results. Finally, in the next graph, we analyze the total number of downloads done for the different network sizes.

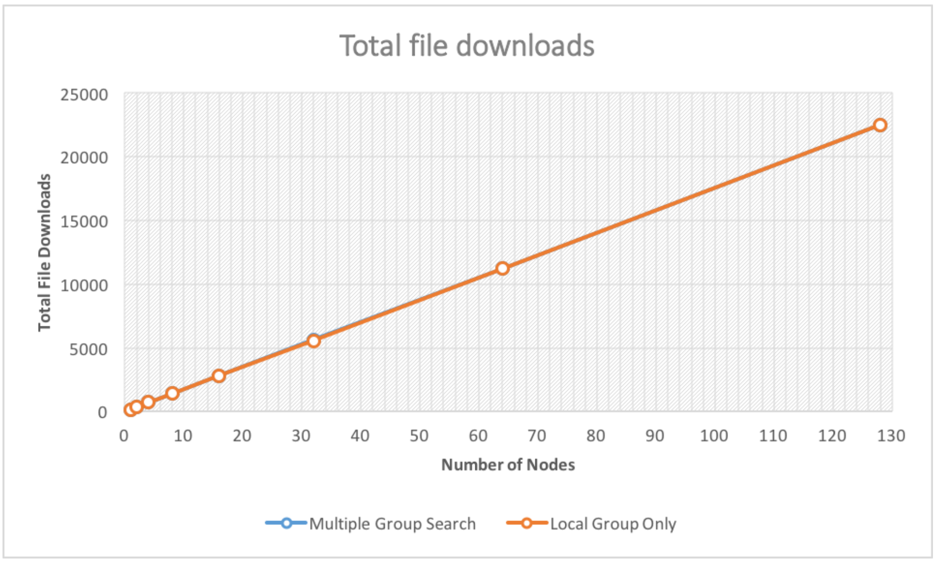


Figure 5.30: Total number of downloads in the Peer-To-Peer overlay

The graph in Figure 5.30 shows that in this implementation there is no loss of performance when the network grows. This can be stated due to the fact that the tendency curve displayed follows and exponential behavior up to 128 nodes, thus meaning that there are significant delays downloading files which would imply less downloads overall.

5.6 Overlay Comparison and Discussion

In order to further explore the results obtained, in this section, a comparison of the best performing solutions verified in the previous analysis for all relevant parameters.

To compare the reduction of accesses performed to the server by the nodes, the following solutions where chosen to perform the comparison.

Scenario A Peer-To-Peer with tabled approach

Scenario B Peer-To-Peer with 2 hops

Scenario C Scatternet with local group search

Scenario D Scatternet with multiple group search

We chose these scenarios since they were all scalable and are possible to implement in our current implementation with real devices, using the same technologies used in the simulation.

Figure 5.31 compares the percentage of local download achieved by each of the selected scenarios.

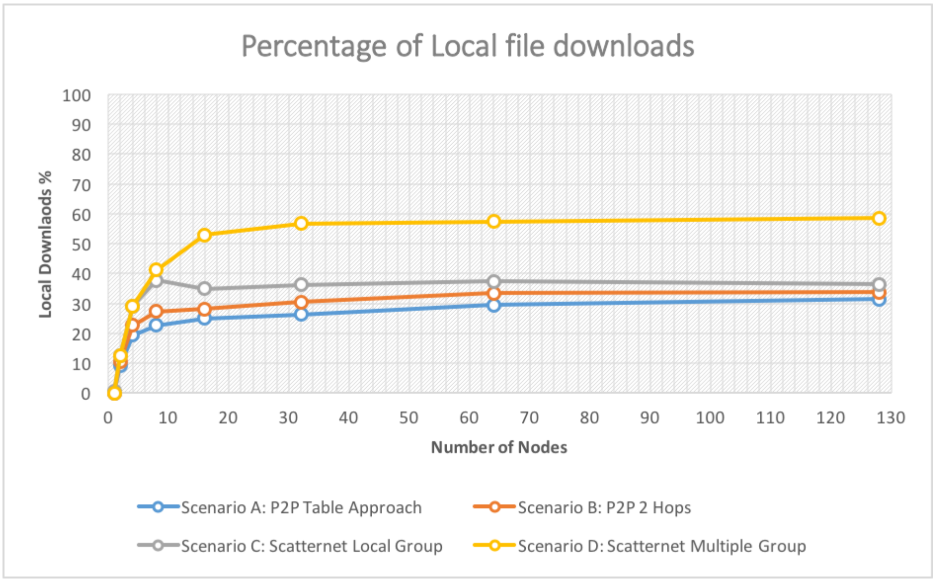


Figure 5.31: Percentage of local downloads.

From this graph we can verify that the best performing overlay to reduce the number of server accesses is the Scatternet, which was expected since we are guarantee to have a denser network than with the Peer-To-Peer approaches.

Figure 5.32 compares the results obtained on the average number of hops each request takes to be answered in the local network.

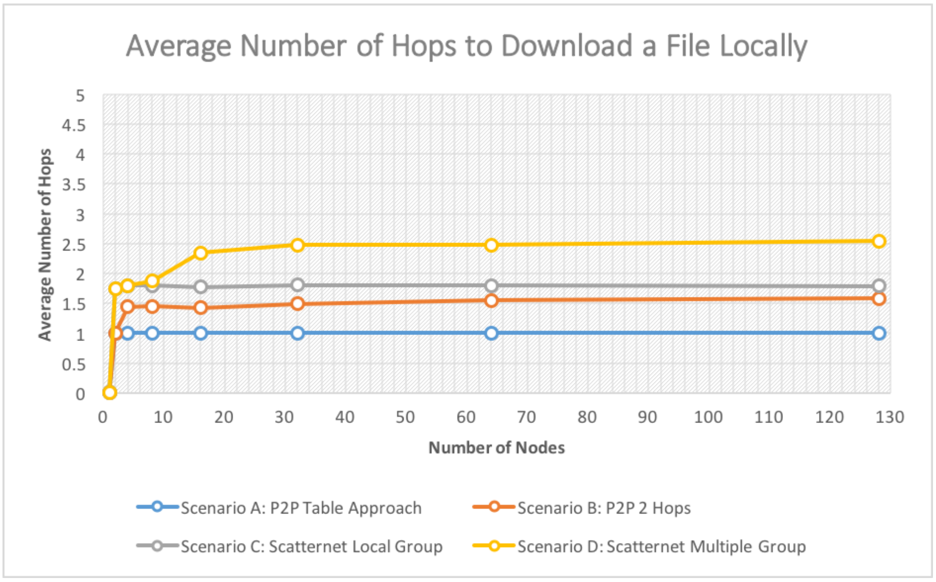


Figure 5.32: Average number of hops.

As expected, the Scatternet with multiple group search has an higher average than both Peer-To-Peer networks. Up to 128 nodes, this did not revealed to be a problem, but if the network grows very large, it may start do show some degradation because the higher number of communications done.

Finally, Figure 5.33 compares the total number of downloads done on each of the selected scenarios.

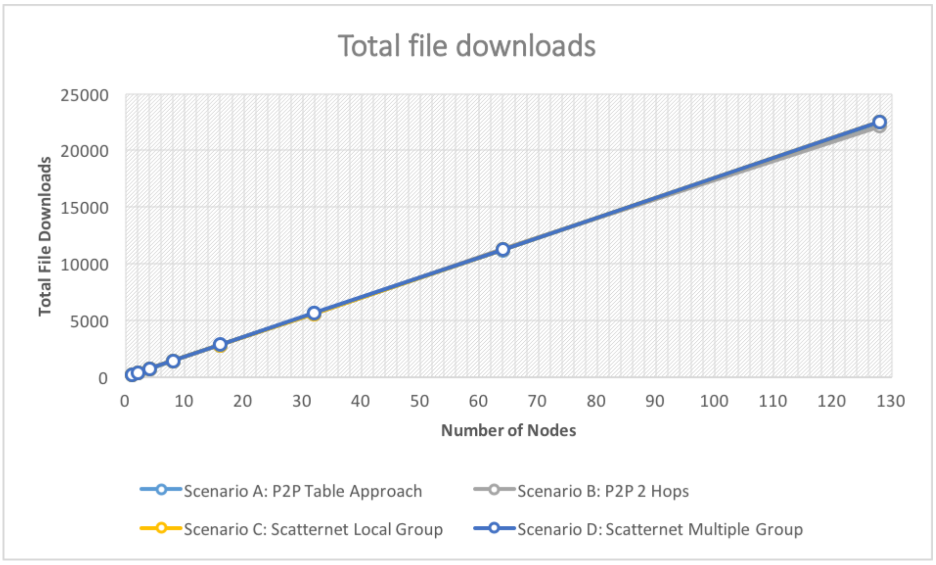


Figure 5.33: Total number of downloads.

We can see that for all cases, the number of downloads was approximately the same, thus their scalability was similar up to 128 nodes.

5.7 Conclusion

From our experiment and its results, we can denote that it is very hard to “have the cake and eat it”. Choices and sacrifices have to be made since each overlay has it pros and cons, and the best solution has to be chosen according to the objective of each case.

Using a star/scatternet based approach it is clear that the amount of messages exchanged, the delays and reduction of server accesses provides better results than both peer-to-peer or ring based approaches. These networks also have less subjective parameters to adjust, such as timeouts, search depth, etc.. But not everything is an advantage. For instance, if we wish a very fault tolerant and churn effect resilient network, a less structured (with more redundant connections) approach should be used. This would make the network formation much simpler and faster, but the consequence would be that the number of local files available in the close neighborhood would be lower when compared to the more structured approach.

From this study, one of the most important conclusions taken is that, the best solution could pass through and hybrid approach. Using the motivation experiments as an example, it is known that during a match, most supporters will not leave their seat, which means that the churn effect will be reduced. During this period a structured approach could largely benefit the quality of this service. Changing from this structure approach to a more resilient one during the halftime, could mean that since the churn is expected to increase, the network will be more prepared for this problem.

In conclusion, from all the studied overlays, the most appropriate solution is the scatternet since it is the one that reduces the most the amount of server file requests without saturating the network with a large number of messages and it is possible to implement using Bluetooth and Wi-Fi on currently available Android Smartphones and Tablets.

Chapter 6

Discussion

Deploying mobile ad-hoc networks to create crowd-sourcing edge-clouds is, as expected, a complex procedure and lots of research is still required to make them useful computational platforms. In this dissertation we addressed the specific case study of content dissemination in sports venues, one of the key scenarios to be studied under project Hyrax.

From the beginning of our work it was clear that simulation would be needed, not only to facilitate the gathering of results, but mostly due to the difficulty in assembling a large enough experimental setup with actual physical devices. We did a survey of current simulator frameworks, and concluded that the best choice for our work was the NS-3 simulator.

We were interested in trying out several wireless technologies, e.g., Wi-Fi and Bluetooth, to check which would provide better efficiency for the kind of applications we were interested. We instrumented the simulator with data taken from experiments with a small test bed of smartphones (LG G2) and tablets (Google Nexus 9). Since NS-3 does not provide a primitive implementation for TDLS and Wi-Fi Direct we did our own implementation of these technologies withing NS-3, based on available technical documentation, and checked that the observed behavior in the simulations was consistent with that observed in the physical device testbed.

This first simulations led us to conclude that the best approach alternative for our case study would be to use Wi-Fi or Wi-Fi Direct with TDLS enabled.

Next, we wanted to study alternative ways of organizing nodes in the edge-networks so that we would get the best QoS for content delivery. In these experiments, we used Bluetooth to create and maintain the overlays while Wi-Fi was used to do the actual video transfers. Bluetooth was chosen due to its low power requirements and low bandwidth which is nevertheless more than enough to exchange messages to maintain a mesh. We tested four overlays: ring, star, peer-to-peer and scatternet. The simulations showed that the ring and star overlays are either not feasible or do not scale. On the other hand the peer-to-peer and scatternet overlays do scale but each has some pros and cons. We concluded that, for the sports venue case study the most adequate solution would be a scatternet, since we want to reduce server and AP load as much as possible.

6.1 Future Work

The research presented in this dissertation constitutes a first reconnaissance of the problems involved in developing efficient edge-networks for crowd-sourcing content dissemination. While we got interesting results that point to specific technologies and overlays as the best overall solutions for our case study, there are many alternative scenarios that can be tested and that may be of interest. We describe some of these in the following paragraphs.

The scenarios we tested in chapter 4 can be further explored by considering hierarchical combinations in which, e.g., Wi-Fi Direct is used in conjunction with fixed servers or with multiple APs. In terms of overlays, many other topologies have been proposed in the literature that might be relevant for our work, e.g., Distributed Hash Tables. We could also increase the size of the networks we are simulating to see how these alternatives scale, although it is not clear whether this is relevant for real world deployments as there might be hard limits on the number of devices supported.

In terms of low level protocol used for video dissemination we could alternatively use UDP rather than TCP for increased efficiency and lower energy consumption. To guarantee acceptable QoS levels we would have to implement ourselves mechanisms to ensure it. It is not clear what are the trade-offs in this approach.

References

[1] Bluetooth Technology Website. `http://www.bluetooth.com/`. Online; accessed June 2015.

[2] Guifi home. `https://guifi.net`. Online; accessed June 2015.

[3] Hyrax homepage. `http://hyrax.dcc.fc.up.pt`. Online; accessed June 2015.

[4] Nyc mesh home. `https://nycmesh.net`. Online; accessed June 2015.

[5] Opengarden home. `https://opengarden.com`. Online; accessed June 2015.

[6] Span project home. `https://github.com/ProjectSPAN`. Online; accessed June 2015.

[7] Wi-Fi Alliance homepage. `http://www.wi-fi.org`. Online; accessed June 2015.

[8] YinzCam Homepage. `http://www.yinzcam.com`. Online; accessed June 2015.

[9] Optimized link state routing protocol (olsr), 2003.

[10] Wi-fi certified wi-fi direct^{TM}: Personal, portable wi-fi® to connect devices anywhere, any time. Technical report, Wi-Fi Alliance, 2010.

[11] Wi-fi certified^{TM} tdls: Easy-to-use, security-protected direct links to improve performance of wi-fi® devices. Technical report, Wi-Fi Alliance, 2012.

[12] WDSim: WiFi Direct Simulator - Developers Guide, 2013.

[13] ZigBee Alliance. Zigbee specification, 2006.

[14] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[15] Donn B Baker, David R Johnson, and Ralph E Sipple. Multi-cast digital video data server using synchronization groups, December 10 1996. US Patent 5,583,561.

[16] Sourangsu Banerji and Rahul Singha Chowdhury. On ieee 802.11: Wireless lan technology. *arXiv preprint arXiv:1307.2661*, 2013.

[17] Sardar M Bilalb, Mazliza Othmana, et al. A performance comparison of network simulators for wireless networks. *arXiv preprint arXiv:1307.4129*, 2013.

[18] Athanassios Boulis. Castalia: Revealing pitfalls in designing distributed algorithms in wsn. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, SenSys '07, pages 407–408, New York, NY, USA, 2007. ACM.

[19] Daniel Camps-Mur, Andres Garcia-Saavedra, and Pablo Serrano. Device-to-device communications with wi-fi direct: overview and experimentation. *Wireless Communications, IEEE*, 20(3):96–104, 2013.

[20] Saloua Chettibi and M Benmohamed. A multipath energy-aware on demand source routing protocol for mobile ad-hoc networks. *arXiv preprint arXiv:0902.4572*, 2009.

[21] J Chroboczek. The babel routing protocol, rfc 6126. *Quagga Routing Software Suite, GPL licensed*, 2011.

[22] eMarketer. 2 billion consumers worldwide to get smart(phones) by 2016. `http://www.emarketer.com/Article/2-Billion-Consumers-Worldwide-Smartphones-by-2016/1011694`. Online; accessed June 2015.

[23] Matthew Gast. *802.11 wireless networks: the definitive guide.* " O'Reilly Media, Inc.", 2005.

[24] Nishant Gupta and Samir R Das. Energy-aware on-demand routing for mobile ad hoc networks. In *Distributed Computing*, pages 164–173. Springer, 2002.

[25] Z.J. Haas. A new routing protocol for the reconfigurable wireless networks. In *Universal Personal Communications Record, 1997. Conference Record., 1997 IEEE 6th International Conference on*, volume 2, pages 562–566 vol.2, Oct 1997.

[26] 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.

[27] Huawei. Huawei Agile Stadium Solution. `http://e.huawei.com/en/marketing-material/global/solutions/agile_network/campus/hw_337341`, 2015. Brochure.

[28] IEEE Standards Association. IEEE 802.11$^{\text{TM}}$: Wireless LANs. `http://standards.ieee.org/about/get/802/802.11.html`. Online; accessed June 2015.

[29] IEEE Standards Association. IEEE 802.15$^{\text{TM}}$: WIRELESS PERSONAL AREA NETWORKS (PANs). `http://standards.ieee.org/about/get/802/802.15.html`. Online; accessed June 2015.

[30] IEEE Standards Association. IEEE 802.16$^{\text{TM}}$: BROADBAND WIRELESS METROPOLITAN AREA NETWORKS (MANs). `http://standards.ieee.org/about/get/802/802.16.html`. Online; accessed June 2015.

[31] Alberto Martínez Illán. Medium and mobility behaviour insertion for 802.11 emulated networks. Master's thesis, Universitat Politècnica de Catalunya. BarcelonaTech, September 2013.

[32] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[33] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. Klein Haneveld, T. E. V. Parker, O. W. Visser, H. S. Lichte, and S. Valentin. Simulating wireless and mobile networks in omnet++ the mixim vision. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 71:1–71:8, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[34] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011.

[35] Axel Neumann, Corinna Aichele, Marek Lindner, and Simon Wunderlich. Better approach to mobile ad-hoc networking (batman). *IETF draft, October*, 2008.

[36] Navid Nikaein, Christian Bonnet, and Neda Nikaein. Harp - hybrid ad hoc routing protocol. In *Proceedings of International Symposium on Telecommunications, IST*, Tehran, Iran, 2001.

[37] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing, 2003.

[38] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *SIGCOMM Comput. Commun. Rev.*, 24(4):234–244, October 1994.

[39] Venugopalan Ramasubramanian, Zygmunt J. Haas, and Emin Gün Sirer. Sharp: A hybrid adaptive routing protocol for mobile ad hoc networks. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing*, MobiHoc '03, pages 303–314, New York, NY, USA, 2003. ACM.

[40] Network Simulator. ns-2, 1989.

[41] Stoker. Manet manager. `https://play.google.com/store/apps/details?id=org.span`. Online; accessed June 2015.

[42] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 60:1–60:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[43] Mário Véstias. *Redes Cisco Para Profissionais.* FCA, 6ª edição edition, Abril 2009.

[44] Elias Weingartner, Hendrik Vom Lehn, and Klaus Wehrle. A performance comparison of recent network simulators. In *Communications, 2009. ICC'09. IEEE International Conference on*, pages 1–5. IEEE, 2009.

[45] Linux Wirelesss. mac80211_hwsim Driver. `https://wireless.wiki.kernel.org/en/users/drivers/mac80211_hwsim`. Online; accessed June 2015.