TÉCNICO LISBOA

Master Thesis

P2P Communication in Android Devices: Web Access over an Ad Hoc Network

Tomás Falcato Costa

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Advisor(s)/Supervisor(s): Prof. António Grilo
Prof. Paulo Pereira

Examination Committee

Chairperson: Prof./Dr. Lorem Ipsum
Advisor: Prof./Dr. Lorem Ipsum
Members of the Committee: Prof./Dr. Lorem Ipsum
Prof./Dr. Lorem Ipsum

October 2017

Acknowledgments

Abstract

Ad hoc networks are emerging as a possible solution to offload traffic from the infrastructure network, or to help disseminate messages when the infrastructure network is not accessible. However, most existing frameworks only provide single-hop or broadcast message transfers, limiting the possible uses of the ad hoc network. This thesis proposes a detailed framework, with a routing protocol, which is able to establish an ad hoc network allowing users to exchange data in a unicast, multi-hop manner. The developed framework can be used in Android applications for different purposes. It currently establishes Bluetooth connections, but is easily adapted to use other communication technologies. Then, a prototype peer-to-peer Android application is developed, based on the proposed framework, which enables users to real-time web browse without an active Internet connection, where web pages may be exchanged in a multi-hop manner. The prototype was tested in experimental settings. The results show that the proposed framework and developed application provide an acceptable quality of service, that could be further increased using certain technological advances.

Keywords: Ad hoc network; Bluetooth; Wi-Fi Direct; Android; Peer-to-peer application

Resumo

Keywords: Comunicação descentralizada, Wi-Fi Direct, Wi-Fi P2P, Redes Ad hoc, Topologia em Malha, Android.

Contents

List of Tables

List of Figures

Acronyms

ACK Acknowledge Packet. 24

AES Advanced Encryption Standard. 66

AODV Ad hoc On-Demand Distance Vector. 10, 11, 31

AP Access Point. 8, 9, 12, 13, 17

BLE Bluetooth Low Energy. 14, 16, 26, 30, 65

BS Base Station. 5–7

BSC Base Station Controller. 5

BSS Base Station Subsystem. 9, 11

CRT Content Routing Table. 23

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance. 8, 9

CTS Clear to Send. 8

DHCP Dynamic Host Configuration Protocol. 17, 21

DSDV Destination-Sequenced Distance Vector. 30, 31

DSSS Direct Sequence Spread Spectrum. 6

DTN Delay Tolerant Network. 27

EDGE Enhanced Data Rates for GSM. 6

EPC Evolved Packet Core. 7

ESS Extended Service Set. 9

FDD Frequency Division Duplex. 5–7

FDMA Frequency Division Multiple Access. 5, 6

FHSS Frequency Hop Spread Spectrum. 13

GM Group Member. 12, 17–19, 21–24, 26

GO Group Owner. 12, 17–19, 21–25

GPRS General Packet Radio Service. 6

GSM Global System for Mobile Communications. 5, 6

HetNet Heterogeneous Network. 7

HSPA High Speed Packet Access. 6

HWMP Hybrid Wireless Mesh Protocol. 11

IoT Internet of Things. 14, 16

IP Internet Protocol. 7, 17–24

ISM Industrial, Scientific and Medical. 13, 14

ISP Internet Service Provider. 8

LTE Long-Term Evolution. 6, 7, 22

MAC Medium Access Control. 6–9, 13, 16, 18, 19, 25, 27, 31, 34, 36, 37, 39–42, 44–47, 49, 50

MANET Mobile Ad hoc Network. 22, 23

MAP Mesh AP. 11

MIMO Multiple Input Multiple Output. 6–8, 16

MP Mesh Point. 11

MPP Mesh Portal. 11

MS Mobile Station. 5–7, 9

MSC Mobile Switching Center. 5

NFC Near Field Communication. 15, 16

OFDM Orthogonal Frequency Division Multiplexing. 7, 8

PIT Pending Interest Table. 24

PM Proxy Member. 22

QoS Quality of Service. 51

RFID Radio Frequency Identification. 15

RIP Routing Information Protocol. 36

RNS Radio Network Subsystem. 6

RTS Request to Send. 8

SC-FDMA Single Carrier Frequency Division Multiple Access. 7

SDK Software Developer Kit. 26, 27

SIM Subscriber Identity Module. 6

SSID Service Set Identifier. 12, 17

STA Station. 10, 11

TCP Transmission Control Protocol. 20, 21, 55

TDD Time Division Duplex. 7, 15

TDMA Time Division Multiple Access. 5, 6

UDP User Datagram Protocol. 19, 20

UMTS Universal Mobile Telecommunications System. 6, 7

URL Uniform Resource Locator. 42–46, 49

UTRA UMTS Terrestrial Radio Access. 6

UTRAN UMTS Terrestrial Radio Access Network. 6

UUID Universally Unique Identifier. 33

W-CDMA Wireless Code Division Multiple Access. 6

WLAN Wireless Local Area Network. 8, 11, 13, 14

WPA Wi-Fi Protected Access. 9

WPAN Wireless Personal Area Network. 13, 14, 16

WPS Wi-Fi Protected Setup. 9

Chapter 1

Introduction

1.1 Context

Nowadays the demand for better mobile devices is higher than ever. Mobile phones are an indispensable gadget in today's society. Increasingly demanding application and connectivity requirements bring the need for devices with more capabilities, *e.g.* battery life, memory, persistent storage, Internet access speeds, *etc.* With this evolution of equipment, inevitably, comes an evolution of communication technologies.

Mobile phones, usually communicate between themselves in different ways: via mobile cellular networks, via Internet access, via Bluetooth, *etc.* New communication technologies are appearing at a fast pace and the possibilities for using them to provide new services for the users are endless.

The main communication methods use a limited number of central points, that coordinate the communication process between devices, acting as mediators in the communication channel. However, from this dependence, a question arises: if there is a limited number of central points what happens if a partial or total failure from their part occurs. This question has an answer in device to device communications.

There are many devices available, usually more than one per person, see [11], making the creation of an ad hoc network a big possibility to overcome possible failures with central points or even if one is not within reach of any central point. Despite that, this answer is not a substitute to the existing communication methods. It aims to add more range and robustness to the network and possibly reduce the workload of the infrastructured network, which has a limited capacity.

Due to the reasons just stated, ad hoc communication between devices has been lately a hot topic, with several applications being currently offered to mobile users *e.g.* FireChat and Ueppa!, see Section 2.5. This thesis offers a framework to create applications of this nature. To realise this framework a new application of this kind was implemented which allows users to access web pages using ad hoc links when they are not within range of an access point or base station.

1.2 Problem Statement

Given the context above, the main question is: where can the creation of an ad hoc network be of use to the everyday tasks people perform on their mobile devices. There are many answers to this question, thus the difficulty of choosing a relevant topic. Much of the work currently being done focus on chat applications, where messages are transmitted via an ad hoc network. Bearing this in mind, this thesis takes a further step and creates a framework to form an ad hoc network capable of transmitting packets between devices.

A solution is proposed to solve the inability of devices to access web pages, through a peer-to-peer application, even where there is the possibility of indirect Internet access, via communication with other devices and not directly with the infrastructure. The Android hotspot is an existing service that allows devices to share their Internet with the surrounding peers. However, this only works if a device is within immediate range of the hotspotting device, reducing the size and reach of the created network. In the solution proposed, the devices are able to reach the Internet by sending their requests to their immediate peers, who will forward them to their neighbors until the destination is reached, travelling more than one hop[1], see the example of Figure 1.1.

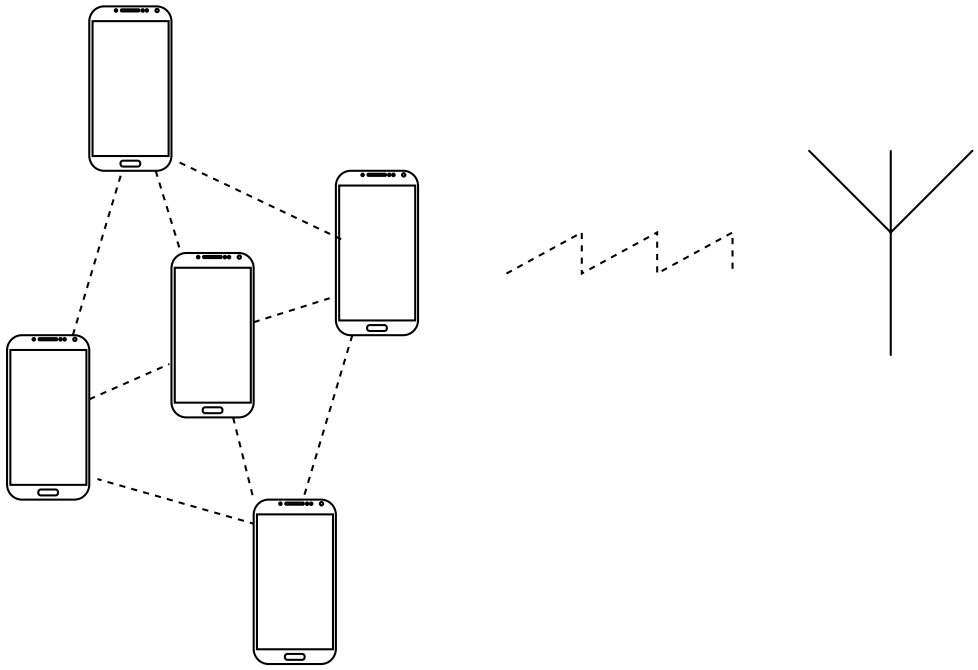


Figure 1.1: Example of an ad hoc network.

It is important to note that this work does not have the purpose of replacing the existing communication infrastructure, but is, in fact, trying to complement it.

1.3 Objectives

This thesis will pursue two main objectives:

1. Developing a framework to create of a decentralized ad hoc network, where packets are transmitted between devices. Proving that, with the current unmodified Android versions and the Bluetooth technology, the creation of a network of that nature is feasible.

   To materialize the framework an application that implements this framework and exchanges web pages between devices is created. The application will create a solid ad hoc network. After the creation is complete the application will provide the logic to correctly manage the web pages request throughout the network, as well as their correct delivery. This application will then be submitted to a series of tests to comprehend where it is more vulnerable and where it is more robust.

[1] One hop is the distance of the communication between a device and its immediate peers. Each time the message travels between a device and its peers a hop is added to the message path.

2. The second objective will be to assess the advantages of migrating this application from Bluetooth to Wi-Fi Direct and what changes need to be made to the current Android versions to accommodate this migration. The advantages and disadvantages of Wi-Fi Direct in comparison to Bluetooth will be compared in the scope of the created framework. Conclusions will be drawn from a series of tests to compare both technologies. Also, a description of the obstacles, present in the current Android devices, preventing the development of this application using Wi-Fi Direct instead will be presented.

This thesis will not provide a market product, thus it disregards some aspects of what would be to expect from a full consumer ready application. Security is not developed in this solution, although some ideas are given on how it can be provided.

1.4 Contributions

As mentioned before the created framework has a huge amount of possibilities. The purpose of this thesis is not to limit these possibilities to the transfer of web pages. It is to provide a simple to use developer kit that can be extended easily to exchange any message format, from web pages to beacon messages.

This is an open source framework and it is hosted in a GitHub repository[2]. In here the full application code will be presented with the necessary comments to complement the description made in Chapter 3. By using the provided mechanisms developers can made the necessary modifications to the code in order to achieve different goals, *e.g.* the creation of a peer-to-peer chat application.

Also, since the transfer of web pages is a complicated process and not much information is available on it, this application has a double usefulness, since it also demonstrates how to exchanges large files between devices.

A set of study tests on the advantages and disadvantages of Bluetooth and Wi-Fi Direct is presented in an easy and succinct fashion. Finally, several study tests on a not so common environment, a Bluetooth peer-to-peer application, are provided, revealing this technology's performance in an Android application.

1.5 Structure of the Thesis

The rest of this thesis is organized as follows:

- Chapter 2 will begin with a theoretical introduction of the different wireless communication technologies, analysing their features, advantages and disadvantages. This aims to provide the needed background to understand the technologies used in this thesis and the choices made along its developments. There will be an analysis on the Android's implementation of Bluetooth and Wi-Fi Direct, as well as some of the possibilities it may present, such as ad hoc networking and multi-hop routing. Finally, an overview of some ad hoc networking applications in Android will be given, describing their features and technologies used.

- Chapter 3 will contain the implementation of both framework and application. It will begin by a description of the steps taken to decide important parts of the framework, such as technologies and routing protocols used. The implemented network creation and communication protocols are explained, providing a better understanding of the framework. Lastly, the materialization of the

[2]To download the code of the application clone the following repository: `https://github.com/Falcato/ThesisApp.git`.

framework, the peer-to-peer application to exchange web page is described, along with its features and overall packet exchanges.

- Chapter 4 will provide a theoretical and empiric evaluation of Bluetooth and Wi-Fi Direct, assessing how both technologies fare in certain aspects, useful for the proposed framework. Furthermore, several experiments will be performed on the developed application, in order to point its overall performance, strong and weak points.
- Chapter 5 will contain a final summary of what was developed and accomplish in this thesis will be presented. The possible future work to be developed in the proposed framework and application will be discussed.

Chapter 2

State of the Art

2.1 Communication Technologies Supported by Mobile Devices

2.1.1 Mobile Networks Technologies

The first form of communication on mobile devices where the mobile cellular telecommunications provided by the Public Land Mobile Network. At first they did not have so many features as we know them now, they were limited to basic voice calls and short text messages. As devices became more sophisticated so did mobile networks, including new features, such as Internet connections and device to device communication.

Mobile networks have become common place, *i.e.*, people make millions of phone calls and text messages everyday, using the service providers' Base Stations (BSs) to enter a network, where their message/phone call is being routed to its destination. This said, it is important in the scope of this work to have some understanding on how devices communicate with each other using these mobile communication standards.

In this subsection we will briefly present the existing standards for mobile cellular networks and their evolution, passing from 2G, 3G and 4G, emphasizing this last one.

2.1.1.1 2G: GSM

Global System for Mobile Communications (GSM) is a standard, created by European Telecommunication Standards Institute, to describe second generation cellular networks. These networks differ from the first generation due to the fact that they were no longer analog, as in 1G, and became digital, allowing for voice as well as text transfer.

GSM's architecture can be seen as hierarchical, with components ranging from Mobile Stations (MSs) to Mobile Switching Centers (MSCs). MSs, the devices, have a unique number, with which a BS can identify each one of the MSs it controls. A Base Station Controller (BSC) controls multiple BSs to allocate radio channels, manage call handover between BSs and control their power levels, in order to avoid muffling the transmission of other MSs. Finally, a MSC, in charge of multiple BSs connects to a Gateway MSC where mobile registration and authentication are made.

GSM uses the air interface to transfer information, being a wireless way of communication, specifically, it uses Frequency Division Duplex (FDD), to separate the uplink and downlink frequencies, 890-915MHz and 935-960MHz, respectively. Then divides each block of frequencies into smaller channels, 125 channels of 200kHz each, using Frequency Division Multiple Access (FDMA). In each FDMA channel it's given a time slot for each MS to use, using Time Division Multiple Access (TDMA). Using this

methodology for medium access, GSM allows for a data rate of 9.6kbps per user, after enctryption and error control overhead.

GSM's main technologies are voice communications, Subscriber Identity Module (SIM) authentication, encryption and accounting information, handover, enabling MSs to move and connect to a different BS maintaining the service and SMS (Short Message Service), allowing for text transfer up to 160 characters, sent to one or multiple destinations.

In order to improve GSM, General Packet Radio Service (GPRS) was introduced, also known as the 2.5G networks, adding two new elements to the previous GSM architecture, a service support node for security, mobility and access control, a gateway support node for establishing connections to external packet switched networks. Although not much improvement on data rate was made on GPRS, soon came Enhanced Data Rates for GSM (EDGE), which combined GPRS with different modulations, improving the spectral efficiency of each channel and allowing for data rates up to 384kbps.

GSM requires heavy resource planning, *i.e.*, frequency and time planning and slot assignment, meaning each user has a dedicated time and frequency and thus the number of users in a cell does not influence the cell size.

2.1.1.2 3G: UMTS

Universal Mobile Telecommunications System (UMTS) was the natural 3G evolution of the GSM/GPRS netowrk. It used the previously created GPRS architecture and improved it using different Medium Access Control (MAC) techniques to improve even further spectral efficiency. The architecture of UMTS is divided into radio access network, UMTS Terrestrial Radio Access Network (UTRAN), in charge of managing cell-level mobility, and Radio Network Subsystem (RNS) and air interface, UMTS Terrestrial Radio Access (UTRA), similar to GPRS. Now UTRAN controls multiple RNSs, who is responsible for handover decisions. The UMTS network operates in parallel with the previously established GSM/GPRS network.

In UMTS transmission is made over two 5MHz FDD channels, using Direct Sequence Spread Spectrum (DSSS), improving both the data rate and security of transmissions. Wireless Code Division Multiple Access (W-CDMA) is now used instead of FDMA and TDMA, each user has a chipping sequence with which messages are encoded, in the destination, with the same chipping sequence the reverse process is made and the message is transmitted, allowing for similar data rates as EDGE and users to transmit simultaneously with little interference, depending on the number of users.

UMTS requires heavy power control, because the source can distinguish each user via their chipping sequences, but if one user muffles another user only one message is received in the destination, thus it is needed to control the power with which each user will transmit. This means the more users transmit simultaneously, more interference is created, assuming non ideal conditions, thus having to reduce cell size to compensate for this interference, leading to more complex cell planning.

In order to enchance the data rates of UMTS, High Speed Packet Access (HSPA) was introduced, which is an evolution of W-CDMA, 3.5G networks. This standard improved upling and downlink speeds, by adding higher-order modulation, *e.g.*, 16QAM or 64QAM, and a more efficient retransmission mechanism in the downlink channel and by allowing parallel transmissions of multiple users, also known as Multiple Input Multiple Output (MIMO), reaching rates up to 168Mbps and 22Mbps, respectively.

2.1.1.3 4G: LTE/LTE-A

Long-Term Evolution (LTE), came to meet the specified requirements in International Mobile Telecommunications-Advanced, issued by ITU-R. But since it did not meet all the requirements to be considered a 4G network, it was considered a 3.9G network. It introduced an exclusively IP-based packet-switching core network,

denominated Evolved Packet Core (EPC), and it targets the increase of quality of service, spectrum efficiency and reduced cost.

EPC introduced new elements to the existing network, a Packet Data Network Gateway, serving as the termination of EPC towards Internet, providing Internet Protocol (IP) services, address allocation, packet inspection and policy enforcement, a Mobility Management Entity, responsible for location tracking, paging, roaming and handover, and a Policy Charging Rules Function to manage the quality of service provided.

LTE uses multiple frequency bands from 700MHz to 2600MHz, with a flexible bandwidth ranging from 1.4MHz to 20MHz, using both FDD, Time Division Duplex (TDD) and a combination of these two methods. This combined with Orthogonal Frequency Division Multiplexing (OFDM) and MIMO, for MAC, allows LTE to reach data rates of 326Mbps for downlink. In uplink a Single Carrier Frequency Division Multiple Access (SC-FDMA) is used allowing for data rates up to 86Mbps. These data rates are considerably higher than the ones reached by UMTS.

In order to further improve data rates on LTE, LTE-Advanced was introduced. This new network meets the requirements to be considered a 4G network, thus it is where 4G networks were actually introduced. Reaching up to 3Gbps for downlink and 1.5Gbps for uplink, Release 10, LTE-Advance immensely improves data rates by using a much wider channel frequency and higher-order MIMO, up to 100MHz, also improving on spectral efficiency.

A new type of networks is also introduced, the Heterogeneous Networks (HetNets), created by deploying a low-power BS at cell edges to enhance network perfomance. Three types of cells are created with this network: micro or pico cells, where a relay node is used to extend the service to other devices. Femto cells, for indoor coverage at home, offices or malls, where a Home eNodeB serves as relay node for devices inside the femto cell.

Considered for 4G LTE-Advanced was the concept of D2D communications, creating direct links between devices within a small area. This technology would enable the linking of devices by using the cellular spectrum, allowing for data to be transferred from one to the other over short distances, but using a direct connection. This form of device to device has a lot of applications, *e.g.*, in disaster scenarios, where the access to the infrastructure is denied, or when the infrastructure is overloaded in *e.g.*, large public events.

4G LTE-Advanced D2D was a feature in Release 12 and brought some benefits, such as reliable and persistent communication, meaning it persists if the LTE network is disrupted, data rates, when the distance to an BS is considerable and interference reduction, by not having to communicate directly with the BS overloading the network. There are of course some issues to be addressed with this communication, such as the authorization and authentication of MSs and the fact that inter-operator communication may not be approved by the different operators, limiting the possible links.

2.1.2 Wi-Fi (IEEE 802.11)

A Wireless Local Area Network (WLAN) is a wireless network that can connect multiple devices to each other within a limited range. WLANs have become very popular in the day to day life of people. Most households have a WLAN deployed so that devices inside and around the premises can have Internet access. The popularity of WLANs is mainly due to the fact that devices do not need to be physically connected to the Access Point (AP) to access the Internet, which removes the costs of cables and the associated infrastructures.

IEEE 802.11 is the *de facto* standard for WLANs and it is commonly known as Wi-Fi. IEEE 802.11 is composed by MAC layer and Physical layer specifications for WLAN implementations, various versions have been released, but the most common are IEEE 802.11a/b/g/n/ac, mainly because they were adopted by mobile and computer manufacturers as the standard to be used in radio communication, *i.e.* wireless.

Unfortunately, the connection speeds are not as high as in wired networks, since the environment plays a big role, *i.e.* if there are obstacles between the AP and devices. Also, the number of devices in the network will affect the data speeds, since the protocol specified for the IEEE 802.11 standard is Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). Although these are disadvantages over wired technologies, there has been done a lot of improvement on the data rates, during the development of wireless networks, reaching up to 6.93Gbps, using MIMO, high-order OFDM and enhanced MAC techniques, see [12] for more information on this topic.

In CSMA/CA the devices check the medium for clearance, *i.e.* if there is no other device transmitting at that time. Nodes attempt to avoid collisions by transmitting the full data only when the medium is clear. Hidden node is still a problem with this MAC method, meaning a node can be transmitting but its transmission is not detected by other devices, creating collisions.

In order to overcome the hidden node problem, Request to Send (RTS)/Clear to Send (CTS) can be used to reserve the access to the shared medium. A control packet RTS is sent by the transmitter, to which the receiver will answer with a CTS. If the channel is clear the packet will be sent immediately, *i.e.* if a CTS was received, else the device will wait a random period of time, named backoff time, before checking if the medium is clear again. When the backoff timer reaches zero, the device will perform the check, if the medium is clear the device will send the packet, otherwise the backoff time is set again. Due to the exponential factor of this backoff time the connections' speeds are limited when multiple users use exhaustively the network channel, whereas in wired connections, like switched Ethernet, traffic management is, typically, done through traffic flow prioritization.

Another main consequence of establishing a WLAN is the security of the communication. In wired networks there is a physical component to security, such as controlled access to the building. In WLANs networks this type of security is not relevant as, for instance, one can enter the network outside the building. Thus security protocols must be implemented to successfully prevent attacks on the communication between devices, such as WPA2 or IEEE 802.11i, see [13] for more information on security protocols.

A WLAN can have three different main modes of operation, infrastructure, ad hoc mode and Wi-Fi P2P. Besides these three main modes, IEEE 802.11s will also be briefly explained due to its characteristics and similarities with this work. In the next subsections these modes will be explained, and provided of pros and cons.

2.1.2.1 Infrastructure

Infrastructure is the most common method, usually deployed and made accessible by a local Internet Service Provider (ISP) or by a Local Area Network. The structure of the network is as follows, there is a wireless AP that manages the various devices on the network and provides the Internet access, this AP

can be either wired, *e.g.* fiber, or wireless connected to network backbone. The AP is responsible for the creation and maintenance of the network, it generates an SSID with which the network will be identified, aswell as a security level, *e.g.* Wi-Fi Protected Access (WPA) or Wi-Fi Protected Setup (WPS).

MSs can then connect to the network via wireless or wired connections. In the wired connections no authentication is required as there is a physical connection and usually higher data throughput is achieved. In wireless connections there is the need of first identifying the network to which the MS wishes to connect and then to proceed according to the security level used by the AP. The access to the network is managed by the AP, the wired connections are, usually, granted priority over the network, whereas wireless connections compete for the usage of the air interface, typically, via a predefined MAC protocol, being the most used CSMA/CA.

An example of the infrastructure mode network layout can be seen in Figure 2.1.

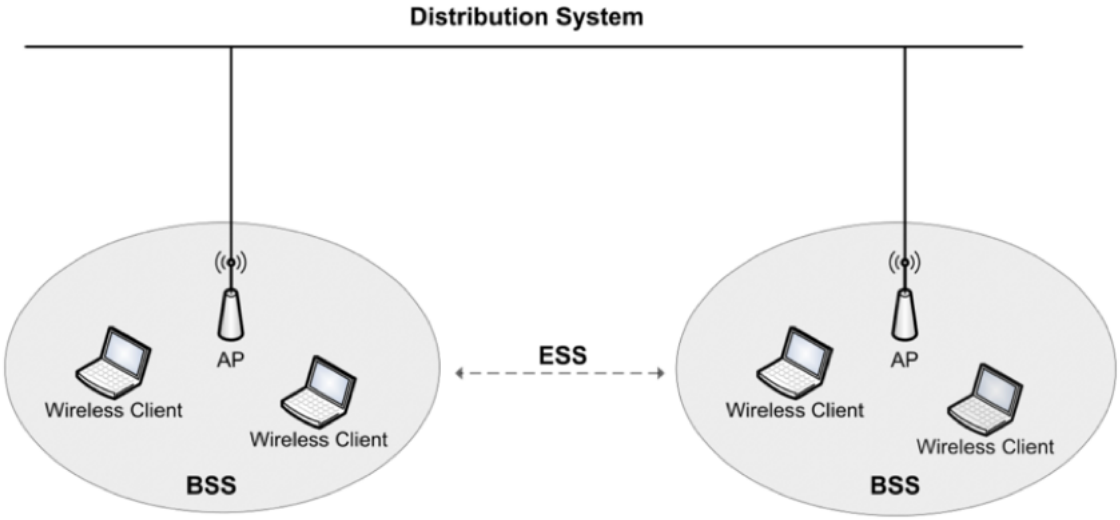


Figure 2.1: Infrastructure network layout (source: www.cse.wustl.edu)

Where each Base Station Subsystem (BSS) is a network and Extended Service Set (ESS) is a set of BSSs that form a single sub network.

The infrastructure mode is ideal if the network as a more permanent character, since the APs are, usually, developed to provide higher-power wireless radios and antennas so that the area covered by the APs is wider. Despite being the most widely deployed method there are some disadvantages associated with this method, for instance, two MSs will not be able to communicate directly even if they reside in the same network, and all their traffic is routed by the AP, which brings another problem: in case of AP failure due to, *e.g.* power failure, software failure, *etc.*, all the network will be compromised and to establish an Internet connection the MSs will have to either connect to another AP or to create the connection by themselves, which brings us to the next subsection.

2.1.2.2 Ad hoc Networks

In the ad hoc mode there is no need for a centralized AP, meaning all the devices can connect to each other if within range. An ad hoc network is slightly different from a Wi-Fi Direct network (WIFI P2P) that will be described in the next section. In ad hoc mode the network is meshed, and all the devices within it are peers, which brings some benefits, *e.g.* the direct communication between devices, without depending on a centralized point connected to the distribution system.

In ad hoc networks with a mesh topology there can be peer-to-peer exchange of traffic. This helps

with the problem of having a centralized point of failure, such as the one present in infrastructure mode. Much like *torrents* files can be transferred by smaller parts and by different providers, by using this method to transfer files, packets, *etc.*, within the network higher data rates can be achieved, since multiple parts of the same file/packet can be sent simultaneously.

An example of the ad hoc mode network with a mesh topology can be seen in Figure 2.2.

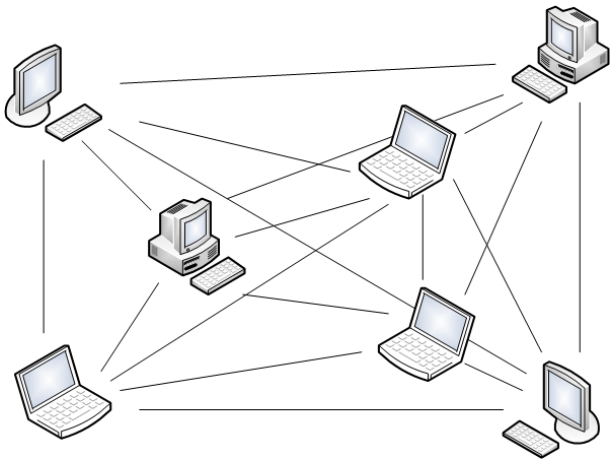


Figure 2.2: Ad hoc network layout (source: monet.postech.ac.kr)

As can be seen in Figure 2.2, one device can have multiple links connecting him to other devices in the network. Although this is a big plus if compared with infrastructure mode, it must be noted that, as said before, a device can connect to other devices if in range, otherwise they will not be able to establish a link and thus communicate. This limitation is due to the lack of a routing protocol in this network mode, so nodes cannot serve as relay for communication between two Stations (STAs), and although it is possible to implement a layer 3 (Network Layer) routing protocol with this mode, such as Ad hoc On-Demand Distance Vector (AODV), it is not intrinsic to this mode. In the next subsection, IEEE 802.11s will be introduced and we will see that it covers this limitation of this ad hoc mode.

Although there are advantages such as not having a centralized point of failure, peer-to-peer file/packet transfer described above, there is an easier setup of the network and the problem described in the infrastructure mode of not existing a direct connection between two STAs is now mitigated as every STA in the network is a peer, there are some disadvantages associated with this network mode, such as the network being more dynamic which brings a lot of changes in the network topology, the interference inherent to all the devices transmitting at the same time to different peers and there is always the scalability problem as more devices in the network mean more connections, which grow exponentially, whereas in the infrastructure mode the connections grow only linearly, so ad hoc networks don't scale well. Also, due to the lack of a routing protocol, the range of the the network will be significantly reduced, as devices do not know which route to forward the packets, in order to reach a certain destination.

Furthermore, the network will not be able to reach the Internet, since devices will communicate between themselves and not with the infrastructure, making the packet exchange limited to the local/cached information stored in the devices, unless if the infrastructure and ad hoc networks are connected through a common device. Finally, the mobility of the devices can make the maintenance of stability of the network a difficult task as links may have to be created and destroyed regularly.

2.1.2.3 IEEE 802.11s (Meshed Network)

IEEE 802.11s is a standard introduced in 2011 which aimed to provide both broadcast and unicast delivery of information. The main difference between the previously described ad hoc mode is that IEEE

802.11s supports multi-hop and implements a layer 2 routing protocol named Hybrid Wireless Mesh Protocol (HWMP).

In this standard, four main types of devices exist: Mesh Points (MPs) who establishes peer links with other MPs, Mesh APs (MAPs) that is a characteristic of MPs which provides BSS services to support communication with STAs, STAs which are devices outside the meshed WLAN and connect to the network via MAP, finally, Mesh Portal (MPP) that is the point at which devices enter and exit the network.

MPs discover potential neighbors based on beacon and probe messages, containing the WLAN Mesh Capability Element, a summary of active protocols and other channel information, and the Mesh ID, that identifies the mesh. The devices are considered to be members of the network upon the establishment of a secure peer link with neighbors within the network. In Figure 2.3, taken from [1], it is possible to visualize an example of the network:

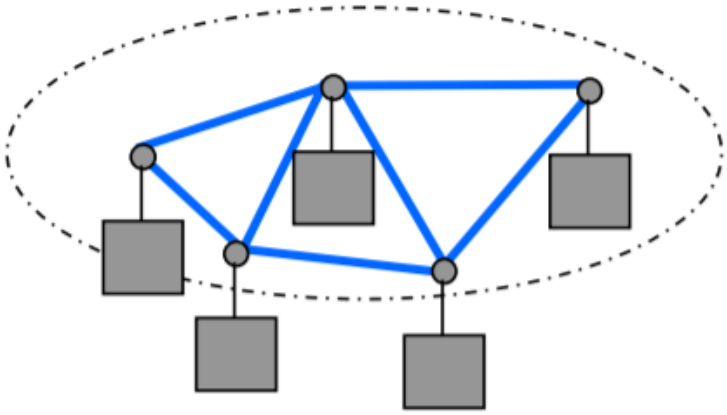


Figure 2.3: IEEE 802.11s network layout (source: [1])

Depending on the number of radio interfaces the devices have, IEEE 802.11s allows for multi-group formation, where each radio interface of each device is assigned to a different group, called Unified Channel Graph.

The routing protocol used by IEEE 802.11s, HWMP, is based on a combination of Radio Metric AODV and tree-based routing, which provides great flexibility in changing environments, great efficiency in fixed mesh deployments, and possible extensibility to metrics other than simple number of hops, such as quality of service, load balancing and power-aware. With this features, HWMP extinguished many of the ad hoc mode flaws, such as lack of routing protocol and thus inability to perform multi-hop transfer of packets and network range.

Although IEEE 802.11s comes with some benefits, pure ad hoc mode still predominates as the peer-to-peer mode, due to its simplicity. There are still some products that make use of the IEEE 802.11s, such as Linux operating system, FreeBSD operating system and Google WiFi routers.

2.1.2.4 Wi-Fi Direct

Wi-Fi Direct is a Wi-Fi standard created by Wi-Fi Alliance. The previously called Wi-Fi P2P, now Wi-Fi Direct, is an innovative way of mobile communication without the dependence of a physical AP. It can be used for different purposes, such as file transfer, Internet browsing, device communication, *etc.*. Wi-Fi Direct assumes an ad hoc topology, meaning the devices are not dependent on one another, but form

a network where all devices share information, hence called peer-to-peer. In Figure 2.4, it is possible to see the difference between traditional infrastructured Wi-Fi (to the right) and Wi-Fi Direct (to the left).

Figure 2.4: Wi-Fi Direct (left) and traditional Wi-Fi (right) network layouts (source: info.tvsideview.sony.net)

Wi-Fi Direct is not dependent on an infrastructure, meaning even without access to a Wi-Fi network it is possible for devices to connect with each other, this because the Wi-Fi Direct enables devices to emit a signal to other devices in the vicinity announcing the possibility of making a connection. Users in the vicinity of the sending device receive an invitation to join a network (Wi-Fi Direct group).

The process of group creation and administration is the most important topic to this work, regarding this technology. Devices can either join existing groups or create new groups, where they will be the administrators, *a.k.a.* Group Owners (GOs) of that particular network. This type of creation forces the Wi-Fi Direct to shape its topology as a star, as is evidenced in Figure 2.4, where there is a central soft AP. It is important to make clear the distinction between a soft and a physical AP: the physical AP usually refers to a physical router, that administrates a network with wired and/or wireless devices, whereas the soft AP can be set up with a Wi-Fi adapter, present in many devices, such as mobile phones, computers, *etc.*.

After the creation of a group, the GO announces to all nearby devices its group, via the Service Discovery protocol, that sends a beacon packet with an Service Set Identifier (SSID), that will be the identifier of the network. Then, the receiving devices can connect to the desired network, by sending information about the device and what type of services it supports. Along with the unique identifier of that device, when received by the GO, the devices become Group Members (GMs) of that network, much like slaves in Bluetooth, see 2.1.3.

In traditional Wi-Fi Direct, the connections are one-to-one or one-to-many, limiting the topology to a star topology, the purpose of this work is to migrate from that star topology to a more dense meshed topology where many-to-many connections are established, and the transfer of data is made faster and without as many relay nodes as in star topology.

The speeds of Wi-Fi Direct are similar to the ones in other Wi-Fi operating modes, reaching up to 250 Mbps. This is the main advantage of Wi-Fi Direct to its direct competitors, such as Bluetooth. As in other wireless technologies, the speed is affected by the environment where the network is inserted, the physical characteristics of the devices and the Wi-Fi physical layer they support, *e.g.*, 802.11a, g or n.

2.1.3 Bluetooth

A Wireless Personal Area Network (WPAN) is type of network where devices are connected wirelessly to each other, based on the standard IEEE 802.15. This definition seems to be quite similar to the one of WLAN, but there is a considerable number of differences between the two. The term personal area network derives from the use that is to be given to such networks, in other words, WPANs are to be deployed in order to connect multiple devices of one's personal area, such as home, office, *etc.*.

Bluetooth is one of the main technologies to implement a WPAN, described above. It uses the 2.4GHz Industrial, Scientific and Medical (ISM) band, it was invented by phone company Ericsson, and is used to connect devices in a short range network.

Devices connect to each other forming *piconets*, which is the term assigned to designate an ad hoc network formed by devices using Bluetooth. Each *piconet* has a master, the device that controls the network, similarly to an AP without providing access to the infrastructured network. Associated to each master there can be up to seven slaves, which are devices that take part in the same *piconet*. Multiple *piconets* can connect between themselves and form a *scatternet*, as seen in Figure 2.5.

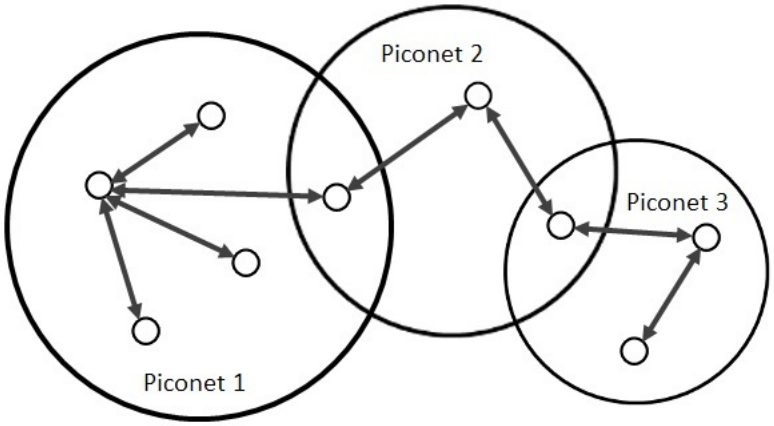


Figure 2.5: Scatternet Layout (adapted from: www.summitdata.com)

Each master is associated with a certain number of slaves, that can participate in more than one piconet, as seen above. These slaves are responsible for coordinating both *piconets*, so that no interference exists. A device can only be a master at one *piconet* at a time thus, the connections between *piconets* must be done at a slave device or at a device assuming both roles, slave in one *piconet* and master in the other one.

Bluetooth uses slow Frequency Hop Spread Spectrum (FHSS) to control the frequency of transmission of each slave, this is done by creating a hopping sequence partially based on the master device's MAC address, and then distributing the sequence to each slave on the *piconet*. Devices connect to the *piconet* by pairing with the master, forming a secure link, the master then controls the access to the medium by deciding which slave will transmit at a certain moment in time. In the *scatternet* case, the data to be transmitted from *piconet* to *piconet* is relayed by the node participating in both networks. The pairing of both *piconets* is similar to the pairing of a master and a new member of the network.

Although several advantages of Bluetooth are clear, such as low power wireless protocol, low transmission headers, ease of set up, multi group information transfer, this protocol still has a lot of downsides. The main current problems with Bluetooth consist in the low range of connections, due to the week signal power being a feature of the technology, the limited number of users that can form a *piconet*, due

to the narrow band used by Bluetooth and, finally, the low data rates of the protocol, which are heavily surpassed by the Wi-Fi standard – see 2.4.

WPAN uses, typically, technologies that allow communication between devices within a certain specific range, usually around 10 meters, making this type of network much smaller than the ones created by WLAN. The most common technology is Bluetooth, although there are several other technologies that are beginning to raise awareness, due to the crescent interest taken in Internet of Things (IoT), such as ZigBee. The used radio band is the 2.4 GHz ISM band, due to its general availability worldwide and its lower cost.

Bluetooth Low Energy (BLE) is the power-version of Bluetooth developed for IoT. The natural power-efficiency of Bluetooth combined with lower energy consumption provides the key factors for devices running for long periods of time without recharging. BLE's key features include: standard wireless protocol, allowing for interoperability across platforms, low idle power consumption and data encryption for security of communications, among others.

BLE achieves data rates similar to classic Bluetooth over the same distance, although the application throughputs are much smaller: 0.27Mbps compared to 0.7-2.1Mbps for classic Bluetooth. The spectrum range of BLE is the same as in classic Bluetooth, but the channels are two times wider than in Classic, consequently, there is half the number of channels. The main difference between the two lies in the power consumption and peak current consumption, 0.01-0.5W and less than 15mA for BLE, 1W and less than 30mA for classic Bluetooth. The number of slaves in BLE is implementation dependent as opposed to the fixed seven in classic Bluetooth.

BLE is appropriate for networks that rely on the longevity of the battery of the devices, application throughputs are smaller but the information to be sent is, usually, also much less than in classic Bluetooth. One of the biggest limitation of BLE is the inability to transmit voice, whereas classic Bluetooth is able. Despite this and other disadvantages of BLE there are many areas that can benefit from technologies such as this one, *e.g.* healthcare applications, logistic sensors, sports, among others, so it is not a technology that should be overlooked.

2.1.4 NFC (Near Field Communication)

Near Field Communication (NFC) is a short-range high frequency wireless communication technology that allows data transfer between devices over small distances, first introduced by three manufacturers, Sony, Nokia and Philips. Communications maintain interoperability between other different communication methods, such as Bluetooth.

NFC can be used to connect mobile applications with the physical world, *e.g.* home appliances, connect devices through physical proximity, forming a peer-to-peer network, and card emulation, creating a connection to a common infrastructure and allowing some actions on the infrastructure, such as making payments.

Evolved from Radio Frequency Identification (RFID) technology, an NFC chip operates as one part of a wireless link. Once it is activated by another chip, small amounts of data can be exchanged between the pair if within a few centimeters from each other. One of the advantages, compared to other wireless communication technologies, is that NFC does not requires a setup to pair two devices, thus reducing the time and packets exchanged in the transaction, allowing for times up to 1ms. Also, NFC chips run on low amounts of power, making this technology much more power-efficient than other technologies.

The short range of the NFC technology is a disadvantage due to its spacial limitation, but in terms of security, this spacial requirement provides a higher degree of security than Bluetooth. For instance, making NFC relatively secure to use in crowded areas, where other wireless technologies could be impossible to use to transfer sensitive data, such as credit card data.

NFC operates at 13.56MHz using Amplitude Shift Keying as the modulation scheme and TDD for simultaneous receive and transfer of data, achieving data rates up to 424kbps, and although these rates are not impressive, for the amount of data that is sought to exchange using this technology and the lack of necessity for communication setup, NFC provides relatively fast transfer times.

2.1.5 Conclusion

Mobile network technologies have been improving at a fast pace, since the demand for higher speeds is constant. With the evolution of modulation methods to higher-orders and MAC protocols improving spectrum efficiency and number of users in the network without interference, the demand for higher speeds has been successfully answered. 5G networks should focus further on resource optimization and a massively distributed MIMO system.

Wi-Fi technology has many different standards, some being the natural evolution of the others, some serving different purposes, such as IEEE 802.11s. The Wi-Fi infrastructure mode has been constantly updated with better MAC and modulation techniques, allowing for higher data rates and more users on the network. Ad hoc networks have also been evolving being the best candidate to offload some of the traffic in the infrastructure mode, also to achieve smaller, independent networks. Wi-Fi Direct has appeared as a possible method to implement ad hoc networks. Its support is still limited in devices, only allowing for some of the features it can provide. Progresses must be made in order to utilize this technology to its full capabilities.

Bluetooth has had a similar development to the IEEE 802.11 standard, evolving to faster data rates from version to version. Used for smaller networks than Wi-Fi, Bluetooth is widely used to deploy WPANs, now with the concurrence of Wi-Fi Direct, although they can both be used simultaneously. BLE was also a big development in low energy networks, allowing for fast data transfer with low power consumption. Each Bluetooth technology has its utility, and the future focus should be in expanding the number of allowed users and range of the networks.

Finally, NFC provides technology for yet another type of network. This time with a range even smaller than WPANs. It has a lot of applications but it's widely known for its easy and secure usability in transactions. It is being researched by industry giants like Amazon and AliBaba, but there is still much room for improvement, in security, network range, data rates, *etc.*.

Overall we can say that most technologies have met huge improvements in short periods of time, and the tendency is to continue that way. Data rates will continue to grow as higher-level modulation techniques are discovered and new MAC protocols are proposed. More emphasis has been given to smaller device to device networks in later years, has a way to take some load from the infrastructure, or even to for networks relevant to day to day tasks (IoT).

2.2 Bluetooth in Android

For the scope of this work, we are specifically interested in the way Bluetooth and Wi-Fi Direct are implemented in Android devices. Various Bluetooth versions have been developed for Android systems. Bluetooth versions over 4.0 are the currently employed in Android devices thus, this section will mainly focus these versions.

In the following subsections, three use cases of the Bluetooth technology in Android devices will be explained regarding, how to achieve a star network, an ad hoc network and multi-hop routing. When appropriate, analysis of previously developed works will be presented, to further demonstrate how these use cases can be achieved using the Bluetooth technology.

2.2.1 Bluetooth Star Formation

As mentioned in Subsection 2.1.3, Bluetooth is able to achieve a star network by creating *piconets* (see Figure 2.5). A master node manages the communications with the slaves, by coordinating the medium access of the various devices in the *piconet*.

In Android, depending on the device's hardware, it is possible to establish several concurrent Bluetooth connections between devices – see [14]. A master device is not able to connect to multiple *piconets* and to form a *scatternet*, as mentioned in Subsection 2.1.3. As mentioned in [15], the role of a device can not be altered in the user space (see Figure 2.6), being defined in the baseband and link controller layer of the Bluetooth chipset. This inability to choose whether a device acts as a master or slave may impact the *piconet* formation process. In Android, it is common to name the Bluetooth roles of a device, server and client, referring to the device that accepted and requested the connection, respectively. This nomenclature may be misleading, as the server may not be the master and the client may not be the slave.

Several applications have been developed, to establish a Bluetooth *piconet* in Android devices[1]. However, these applications assume the device is able to maintain seven active connections which, as mentioned above, depends on the device's Bluetooth hardware.

To create a *piconet*, seven different Universally Unique Identifiers (UUIDs) are created, one for each peer device. The *piconet* initiator, starts a discovery process, where the nearby devices with Bluetooth on will be discovered, *i.e.*, the device that initiated the discovery will have access to the nearby device's Bluetooth name and MAC address. Once the device finishes the discovery process, it attempts to establish a connection with each peer device. It is important to note that this device may not be the master, as seen before. For each successful connection attempt, the peer count is increased by one, until a maximum of seven devices is reached. For each connected device, a communication socket is opened for data transfer between the pair, establishing a *piconet* or a *scatternet*, depending on the master/slave role assignment.

2.2.2 Ad Hoc Networking

In the Bluetooth standard, ad hoc networking is achieved through the creation of *piconets* and *scatternets*. As mentioned in the previous subsection, a *piconet* uses a central node, the master, and several leaf nodes, the slaves, creating an ad hoc network, where the focal point is the master. When several *piconets* are communicating, several master nodes act as focal points of the network, thus creating a decentralized network.

[1] Two GitHub repositories with Android applications claiming to form a Bluetooth *piconet*: `https://github.com/jonnysgomes/piconet/blob/master/src/com/example/piconet/Piconet.java` and `https://github.com/ilteoood/bluetooth_piconet/tree/master/app/src`.

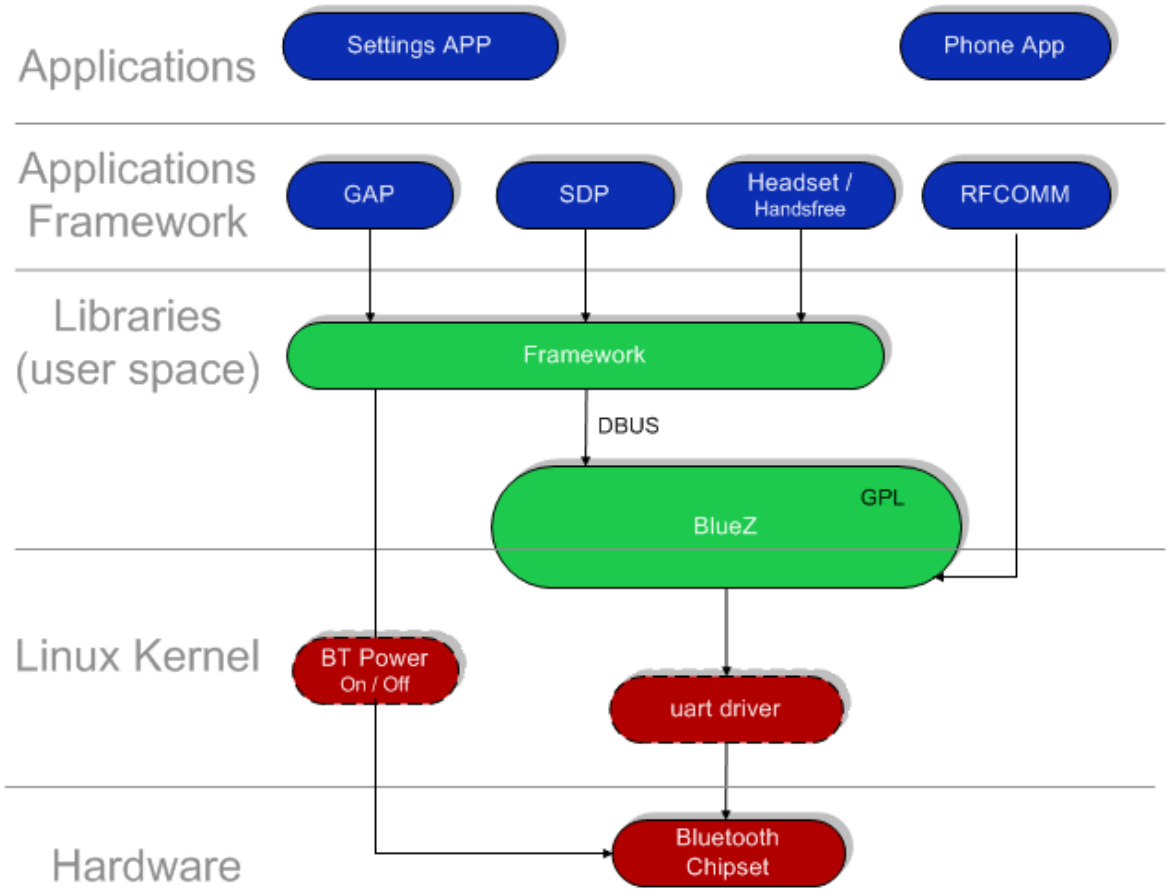


Figure 2.6: Bluetooth protocol stack in Android devices. (source: [2])

In Android, establishing a *piconet* is challenging, due to its lack of features, such as the choice of master/slave roles for a specific node. However, some works already approached this problem (see the previous subsection).

In [3], D. Zhang *et al.* propose a system architecture to deploy a mobile ad hoc social network in Android systems. They establish four layers to this architecture: device, network, community and application layer. To analyse the formation of an ad hoc network using this architecture, the focus will be in the network layer.

In Figure 2.7, the authors of [3], propose a preliminary network model, based on the proposed scheme. It is possible to see the formation of several Mobile Ad hoc Networks (MANETs), connected by a central sub-network with Internet access, resembling the formation of a *scatternet*. In each MANET, mobile devices establish a linear communication topology, meaning that they hold a maximum of two connections, whereas computers may establish meshed networks between themselves and other devices.

It is also stated in [3] that two distinct problems of the Bluetooth technology in Android systems must be overcome: having no ad hoc function and its constraints of limited power, computation and communication capabilities. The authors state that both these limitations of state of the art Android systems are overcome by the proposed architecture, by (1) allowing devices to discover its neighbouring devices and then, creating local communities, as shown in Figure 2.7; (2) establishing local networks within smaller areas, where devices are physically close to each other and by developing a lightweight client to server communication protocol.

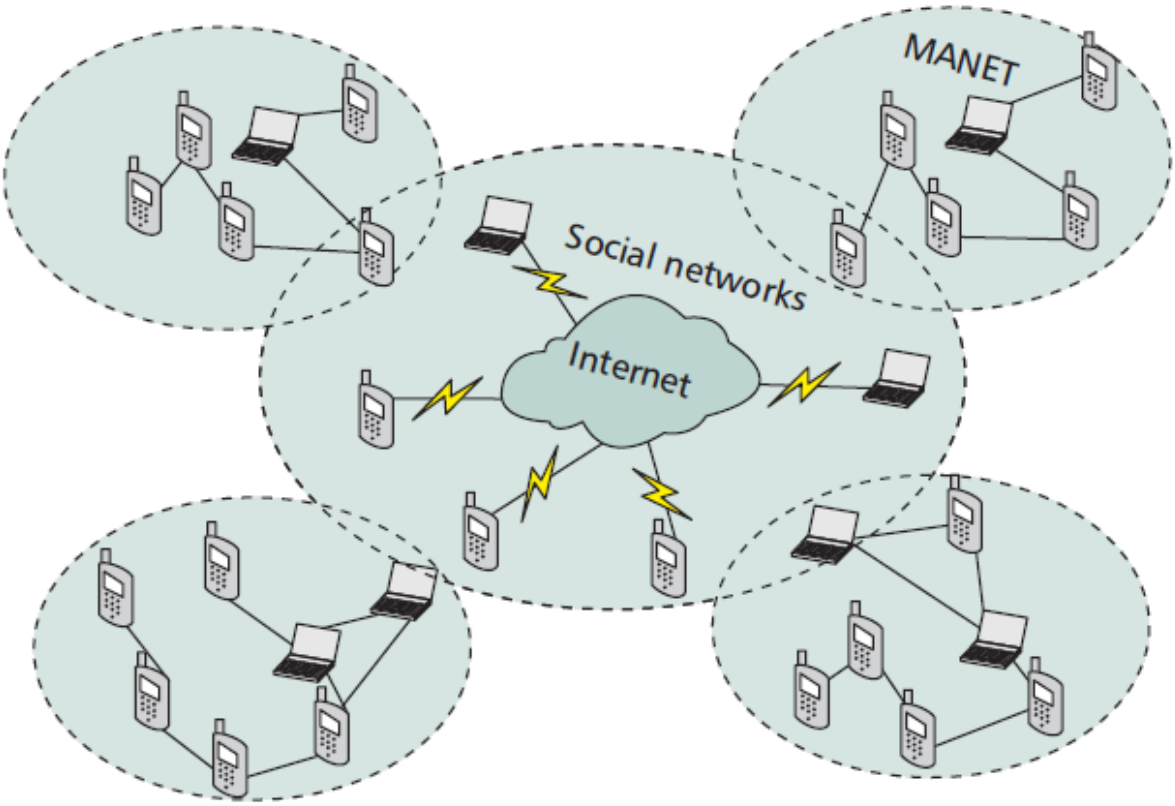


Figure 2.7: Overview of the preliminary network model of the proposed scheme. (source: [3])

In [4], G. Hinojos *et al.* propose a different approach to create an ad hoc network in Android devices using Bluetooth. To achieve ad hoc networking, the authors created what they called *BlueHoc*, a static ad hoc network, composed of a ”boss” and several ”workers”.

In Figure 2.8 an overview of a generic *BlueHoc* network model is shown. The ”boss” node may be connected with up to seven ”workers”, similarly to what happens in a traditional Bluetooth *piconet*. *BlueHoc* networks are static, meaning once a worker connects with a boss it stays connected until the end of the job period.

”Bosses” are able to issue job requests, that are distributed through the ”workers”. When a ”worker” finishes the job, it sends the results back to the ”boss” node. With this mechanism, the authors state that it is possible to achieve parallel computation within a *BlueHoc* network.

In [4], G. Hinojos *et al.* point that, future work may lay in the development and implementation of more elaborate protocols, for ”scatter-gather-broadcast”, substituting the current basic networking architecture.

2.2.3 Multi-Hop Routing

In this subsection, possible approaches to overcome multi-hop routing in Android devices will be presented.

In the previous subsection, the presented works focus the creation of ad hoc networks. However, none of the described works tackles multi-hop routing. In fact, most of the works are focused on establishing local networks, based on proximity between connected devices. For this thesis, multi-hop routing is a limitation that must be overcome, in order to successfully implement the proposed framework and application.

Figure 2.8: Overview of a preliminary network model and data exchange architecture of *BlueHoc*. (source: [4])

As seen in the previous subsection, mobile devices can establish Bluetooth connections, forming network with a linear topology, where each mobile device has up to two concurrent connections. However, routing data through these communication links requires additional logic, to ensure the information reaches the correct destination.

In [5], Y. Wang *et al.*, propose a multi-hop connectivity framework for Android, using Bluetooth and Wi-Fi Direct as communication technologies, called *BWMesh*. This framework discovers and connects neighbour users, using both Bluetooth and Wi-Fi Direct, transmits messages and provides network status to applications using the framework.

BWMesh uses Bluetooth and Wi-Fi Direct's discovery processes to scan for nearby devices. Once two devices are connected, *BWMesh* attempt to maintain their connection as long as possible, for instance by enabling Wi-Fi Direct when a current Bluetooth connection is broken, due to devices being out of Bluetooth communication range.

To communicate between themselves, devices may use Bluetooth or Wi-Fi Direct, depending on the radio interface enabled by the user. However, connection and message transmission may only occur between two Bluetooth or Wi-Fi Direct terminals, cross-technology communication not being possible.

A preliminary simple flooding-based forwarding mechanism is implemented, where a received message is disseminated by the receiver to its nearby peer devices. Also, to unequivocally distinguish devices in a network, the authors propose a device identifier to be an integration of a user-defined semantic name with the intrinsic device serial number. This identification is technology independent, as both Bluetooth and Wi-Fi have different MAC addresses, in an Android device.

In Figure 2.9, the *BWMesh* message format is shown. The exchanged messages are divided in four

segments, each segment being parsed through the corresponding prefix. Prefix (1) *HOP_NUM* denotes the number of experienced hops transferred from the message originator – each time the message is forwarded a unit is added to this segment; (2) *USER_NAME* corresponds to the user-defined semantic name – it is used to associate a device with an easily human readable identifier; (3) *DEVICE_TAG* is the unique identifier of the device in the network; (4) *PAYLOAD_TXT* is the message content to be transmitted.

HOP_NUM	USER_NAME	DEVICE_TAG	PAYLOAD_TXT

Figure 2.9: *BWMesh* message format.

The authors designed and implemented a *BWMesh* based prototype Android application, named *MultiChat*. *MultiChat* enables users to exchange real-time text messages with nearby people in a multi-hop manner, when they are beyond single-hop Wi-Fi Direct range or without an active Internet connection.

In Figure 2.10, a possible network of *MultiChat* is shown. Four users are presented: A, B, C and D. Between A and B, and C and D, a Bluetooth connection is established. Between users B and C a Wi-Fi Direct connection is established.

Once the devices are all connected, the four users are able to exchange text messages in a broadcast manner, similarly to a community chat. Also, although user A is not in communication range of user D, they are able to communicate. Using *MultiChat*, every user has knowledge about the unique identifier and number of hops separating him/her from the remaining users of the network.

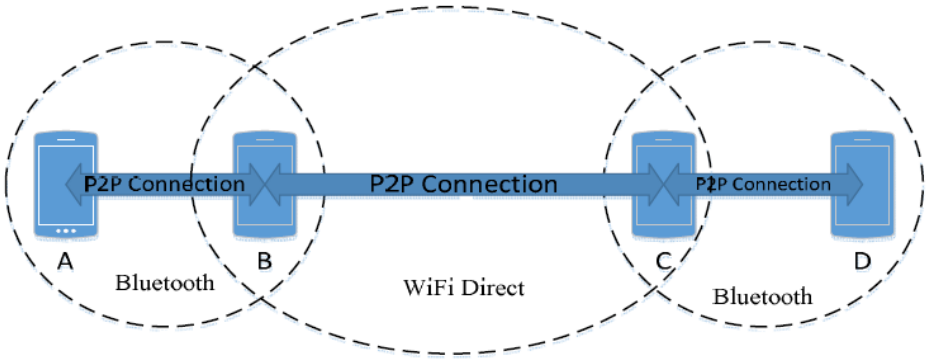


Figure 2.10: Overview of a preliminary network model of *MultiChat*. (source: [5])

The authors state that *BWMesh* may be further improved in the future by:

- Possibly, utilizing Bluetooth insecure communications to avoid the manual pairing of devices in the created network.
- Seeing Wi-Fi Direct's current Android implementation being boosted to the point where no user interaction is necessary, to transfer files between devices.
- Incorporating a more complex, realistic network topology. Allowing the existence of multiple multi-hop routes and destination-specific routing, for instance to allow private messaging.

2.3 Wi-Fi Direct in Android

In this section, the current Wi-Fi Direct Android implementation will be described and analysed. There are small differences depending on the operating systems in which Wi-Fi Direct is being implemented, thus it is not possible to universally describe Wi-Fi Direct with more detail than the one used in Subsection 2.1.2.4.

In the next subsections, the details intrinsic to the Android operating system will be introduced and explained, followed by an in-depth description of various works on how to improve and expand the functionalities of this implementation.

2.3.1 Wi-Fi Direct Star Formation

As previously mentioned in Subsection 2.1.2.4, an ad hoc network is implemented in Wi-Fi Direct, by using a protocol for discovery and connection of the GO with the GMs. The GO functions as a typical Wi-Fi AP, managing the different communications of the GMs.

GOs are not predefined. It is during the group creation that the actual GO is chosen, according to the specified parameters of the protocol, *e.g.* battery percentage. This feature is relevant to manage the vitality of the network as a mesh of groups, since the GOs can be chosen dynamically extending the life of the network.

After the process of the group creation, the GO periodically sends a beacon to advertise the group, enabling other devices to discover and join the group. This advertisement is made in two different ways, either via Wi-Fi Direct, or via typical Wi-Fi. In the first way, devices discover the network via the Wi-Fi Direct discovery protocol, and join using the described set of actions. In the second way, the GO announces the SSID of the network and other devices, also known as legacy clients, connect via infrastructure mode Wi-Fi, using the SSID and password, if set, to identify the GO's network. This leaves us with two different types of clients: legacy and normal. This differentiation will be the key to overcome the lack of multi-group interaction.

In Android devices, IP addresses are predefined according to the function the device performs within the network. The GOs are automatically assigned the following IP address: 192.168.49.1/24, using Dynamic Host Configuration Protocol (DHCP). Whenever a P2P or legacy client connects to the group, DHCP is run again and the clients take an IP address ranging from 192.168.49.2/24 to 192.168.49.254/24, chosen randomly to minimize the chance of conflicts. The GOs are always assigned the same IP addresses, unless they participate in another group as a GM taking the same IP as a regular client – see [6] for a more detailed explanation on this topic.

This technology is still not implemented to the best of its capacity in Android devices. The lack of a routing protocol that can establish multi-group communication, establishing a meshed network is an essential tool to achieve ad hoc networks. The current state of this technology in Android devices is a single group network with one-to-many links established from the GO to the GMs, which limits both the range and scalability of the network. Wi-Fi Alliance states that it is possible to overcome these limitations using stock devices, by creating software to allow for multi-group formation, although it is not standardized in Android's current version 7.0 "Nougat".

In the next section, a collection of developed work will be presented where the different authors propose different methodologies to overcome the lack of this feature in Android devices.

2.3.2 Ad Hoc Networking

C. Casetti *et al.* propose in [6] a process to successfully form a meshed network, by allowing multiple groups to communicate. This proposition is based on stock Android, not requiring any ”root” to be made to the devices, meaning all the actions will be performed in application layer, not envolving any changes in IP addresses or MAC interfaces.

The authors state that multi-group formation can be implemented by taking advantage of both virtual network interfaces of a device, *i.e.*, the Wi-Fi or legacy interface and the Wi-Fi P2P interface, using each one to act as bridge in each group.

This said, upon experimentation, the following scenarios are not feasible in stock Android, due to the inability of creating a custom virtual network:

- a device is the GO of one group and GM in another,
- a device is the GO of two or more groups,
- a device is a GM in two or more groups (non-legacy).

Due to these limitations, the authors propose that a GO be a legacy client in a different group, seen in Figure 2.11:

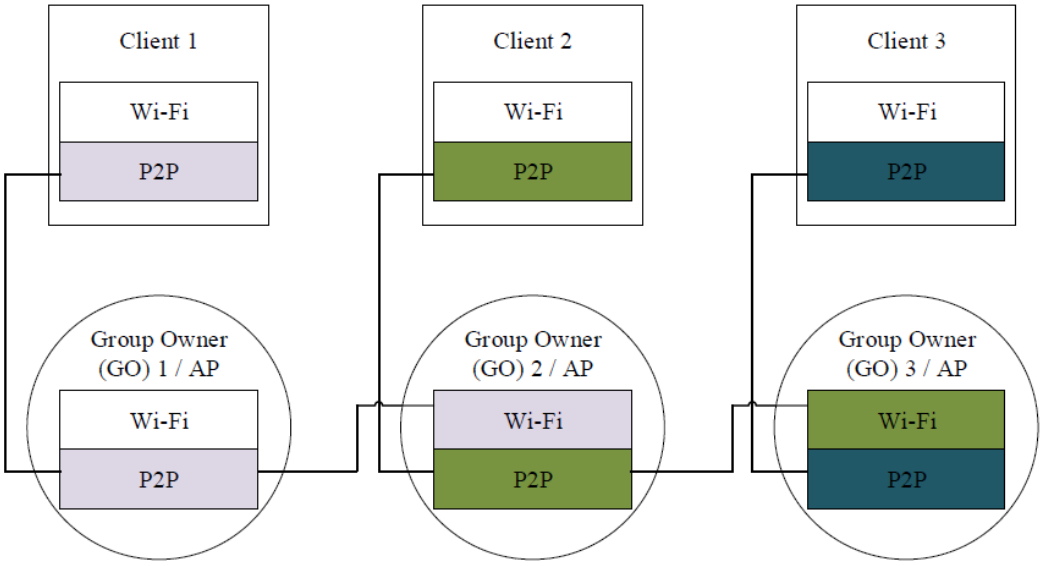


Figure 2.11: Multi-group physical topology with six devices (source: [6])

So, for each GO two network interfaces are enabled, one is the conventional Wi-Fi and the other used for Wi-Fi Direct connection. The IP addresses are assigned according to the previous description.

Two cases are distinguished by the authors: the GO is not connected to any other group as a legacy client, which is the default topology of the network. In this case all connections are feasible, as Wi-Fi Direct has been implemented in order to provide full connectivity among all devices of a single group.

In the second case, the GO is connected to another group as a legacy client as depicted in Figure 2.11 Groups 2 and 3, limiting data transfer to only a subset of D2D data. These limitation are due to two reasons, first the IP conflict of both GOs, who share the same address, 192.168.49.1, making the communication between two adjacent GOs impossible. Secondly, when a GO wants to send a unicast

packet to any client, the packet is sent through the GO's Wi-Fi interface, due to Android's implementation of routing table entries in the GO.

So in this case, client-to-GO communication is allowed since client routing tables list only one interface and there are no conflicts, in GO-to-client direction, bidirectional unicast communication is not allowed. Broadcast communication on the other hand is possible, since it is always sent through the GO's P2P interface. Although when they reach the GO acting as a legacy client they are dropped due to the IP address conflict mentioned above. Finally, client-to-client communication is bidirectional and sent through the client's P2P interface.

It is known from the first case that full connectivity among devices in a single group is allowed, even if one of the GMs is a legacy client and GO of another group. Based on this, the authors introduce the term relay node, which is used to describe this legacy client. The relay node is used to connect two groups. This node is chosen at random, upon the sending of a message from the GO to one of its clients chosen at random among those who do not act as GO in another group. It is important to note that the authors state that this message must be broadcasted to avoid sending it through the Wi-Fi interface, the problem described above in the second case. These clients provide the communication backbone and provide connectivity to all other clients in the group, except from the ones acting as GOs in another group.

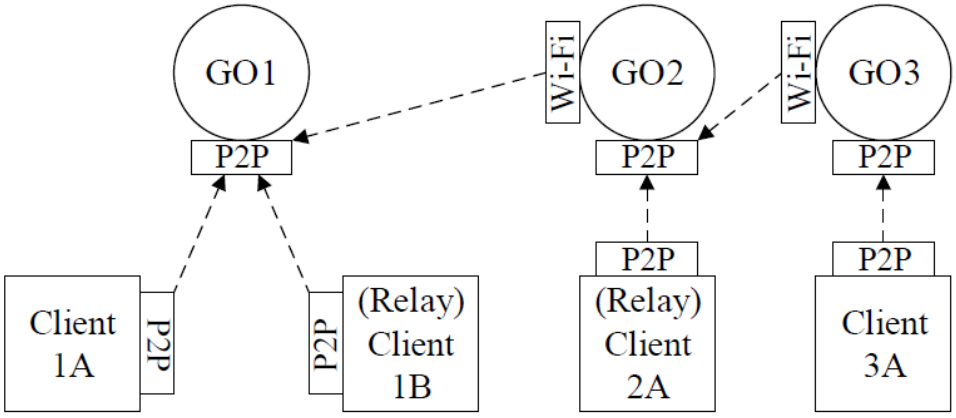


Figure 2.12: Example network topology with 3 Wi-Fi Direct groups. (source: [6])

Take Figure 2.12, where only Wi-Fi Direct connections are represented, for instance. The procedure for Client 1A to send a packet to Client 3A is as follows: Client 1A encapsulates the data in the payload of a unicast User Datagram Protocol (UDP) packet and sends it to the relay Client 1B. The packet will be forwarded by GO1 to Client 1B, at the MAC layer.

The packet is processed at the application layer and the payload is duplicated into a new UDP packet. Sent directly to GO2's standard Wi-Fi interface IP address. At the MAC layer, the packet is sent to GO1, which sends the packet to GO2, via Wi-Fi direct.

The same process is repeated by GO2, but the packet is sent as a broadcast IP packet through GO2's Wi-Fi Direct interface, to relay Client 2A. This client replicates the process of relay Client 1B, sending it to the IP address of GO3's Wi-Fi interface.

Finally, GO3 processes the received packet and sends it to its destination with the correct payload, broadcasting it, similarly to the procedure of GO2.

This mechanism is used following the second case, where the GO is connected to another group as a legacy client. With this mechanism the packet is successfully sent from group to group until its

destination is reached, using a mix of unicast and broadcast communication.

C. Funai *et al.* propose a similar approach in [7]. Their approach is to test two distinct possibilities for multi-group formation, as seen in Figures 2.13 and 2.14, where ”LC” stands for legacy client and refers to clients that use the classic Wi-Fi interface, instead of the P2P interface to connect to a group.

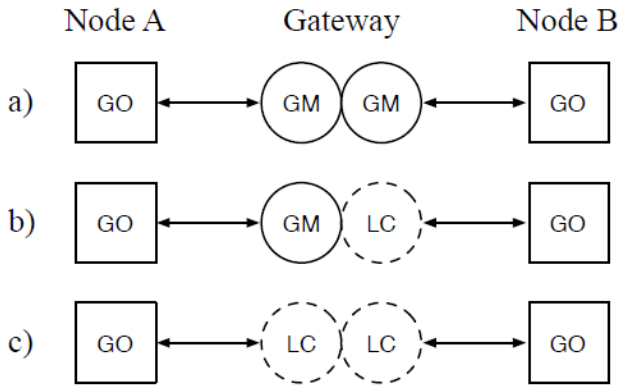


Figure 2.13: Multi-group communication scenarios where the gateway node acts as a client in two groups (source: [7])

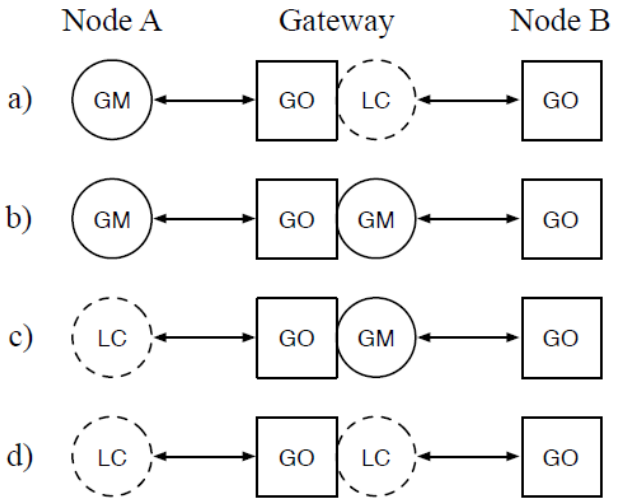


Figure 2.14: Multi-group communication scenarios where the gateway node acts as the GO in one group and as a client in the other (source: [7])

The term gateway node is introduced, referring to the device that connects multiple groups. First by iteratively switching between the different P2P groups, relaying data between them. Secondly by using UDP-based broadcast and a UDP/Transmission Control Protocol (TCP) hybrid solution to achieve multi-group communication. And finally by modifying the source code of the Android operating system.

The authors describe the limitations of stock Android, one of which the multi-group communication must be handled at the application layer and not at the network layer. Actions such as setting IP addresses and managing routing tables cannot be performed without reprogramming the operating system of the device. Another limitation, already referred in [6], is the inability to create virtual network interfaces or multiple virtual MAC entities.

These limitation result in the failure of direct implementation of the test scenarios shown in the figures above. Despite this, the authors state that it is possible to use Wi-Fi Direct functionalities simultaneously with an infrastructure wireless network, reaching the conclusion that the operating system is creating a virtual network interface. But the same interface cannot be connected simultaneously to multiple groups. Thus, scenarios *a)* and *c)* from Figure 2.13 and scenarios *b)* and *c)* from Figure 2.14 are not feasible using simple application layer procedures.

Although it is possible to use both interfaces concurrently and the connection between the groups is successfully established, the experiments from the authors suggest that it is not possible to create a unicast communication to and from the gateway node. For scenario *b)* of Figure 2.13 the gateway node was able to receive data from both groups but was not able to send. For scenario *a)* and *d)* of Figure 2.14 the gateway node was able to communicate with node A but there was no communication with node B. The authors believe this is due to the fact that the DHCP protocol assigns the same IP address to multiple GOs, creating a routing problem, also referred by C. Casetti *et al.* in [6].

Three solutions to this problem are presented. The first is time sharing, which will allow the implementation of any scenario in the figures above. In this solution, the gateway node is alternatively connecting between groups, *i.e.* disconnecting from the current group, scanning for active devices and request to connect to the new group. In the scenarios of Figure 2.13 the gateway node acts as client in both groups, thus neither group has to be destroyed for this switch to occur. Alternatively, in the scenarios of Figure 2.14 one of the groups has to be destroyed, since the gateway node acts as owner for one group and client for the other. The main difference between the different scenarios within each figure is the protocol used to connect and disconnect to a group, *i.e.* Wi-Fi Direct, classic Wi-Fi or a hybrid combination of the two.

A different solution proposed by the authors would allow simultaneous connections between groups. This can only be achieved by using a hybrid combination of the protocol, as already discussed, so only some scenarios will be tested. The authors tested the different topologies with different network sockets, *e.g.* stream, datagram and multicast sockets. These tests showed that when combining a LC/GM, or *vice-versa*, with a multicast socket, the gateway node is able to communicate with both groups simultaneously. Although the multicast socket only encapsulates one-to-many unicast communication, underutilizing the bandwidth of both protocols.

From the authors' experiments, the gateway node is able to receive and send data over the standard Wi-Fi link, while being connected to both groups with a unicast socket. But no data can be routed with the unicast socket over the P2P link. The reason for this is that Android prioritizes standard Wi-Fi links over P2P links. So the *Hybrid* protocol is proposed, where the multicast socket acts as a control channel to change the configuration of the gateway node. The gateway node receives a control message from a GM, it then verifies which type of link it established with the group. In case of a standard Wi-Fi link, the node starts the reception of the data and disconnects from the same group after the reception is finished, creating a TCP connection with the other group to send the data. In the second case, the node is not allowed to receive data, so a notification message is dispatched and the node disconnects from both groups re-connecting with the correct configuration, *i.e.* inverting the link types.

Finally, the authors modified the source code of Android 4.4.2, altering the current implementation of Wi-Fi Direct to assign a unique IP address to each GO, mitigating the routing problem. This allows for a gateway node that uses both interfaces, and acts as GO in one group and as a legacy client in the other.

A. Shahin *et al.* propose an Efficient Multi-group formation and Communication (EMC) protocol for Wi-Fi Direct, in [8]. This protocol allows multi-group formation as the solutions presented before, only this time the protocol has some significant improvements. EMC exploits the battery conditions of the devices in the network to select the GOs of each group and enables the dynamic formation of Wi-Fi

Direct groups. With these features EMC is a very efficient protocol if battery is an essential resource, *e.g.* during an emergency period.

The authors utilized Wi-Fi Direct's service discovery feature to allow devices wishing to form a group to share information on their battery status. The algorithm will then choose the device with a richer energy reserve and elect it as the GO. The rest of the devices can then connect to the created group as GM. Some of these GMs are referred as Proxy Members (PMs) and are the GMs that link a group to another, similar to the definition of bridge and gateway nodes introduced by the other authors. PMs use their standard Wi-Fi interface to join another group, as a legacy client, forming a network topology similar to the one in Figure 2.15.

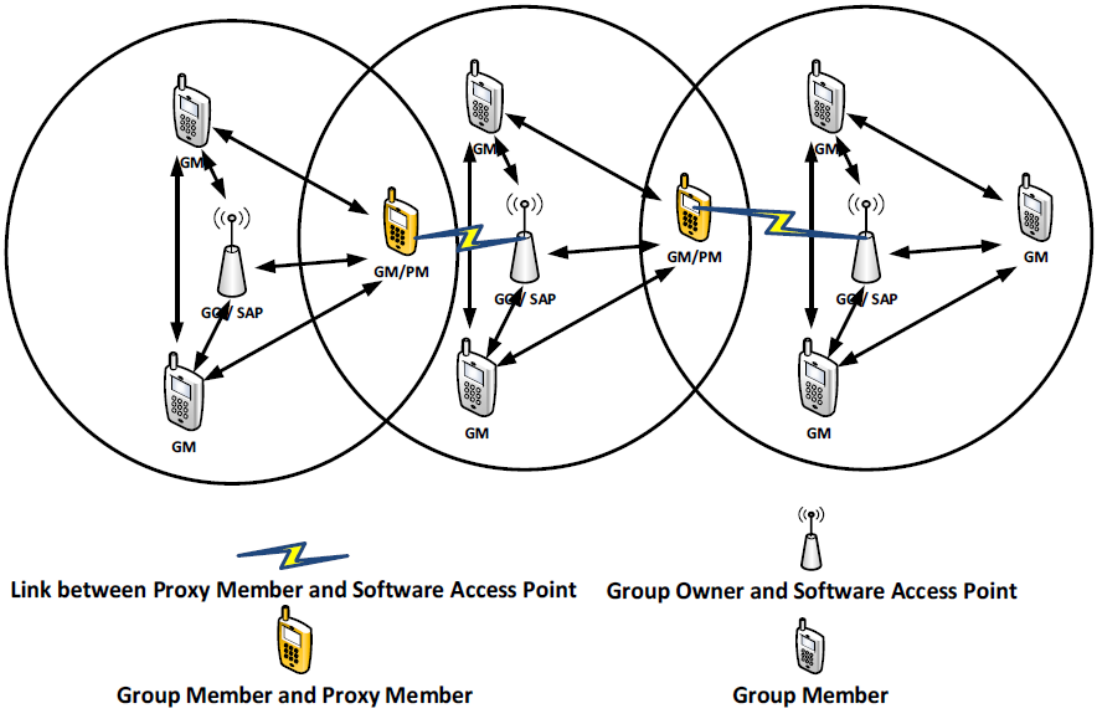


Figure 2.15: Example of a network topology after running EMC (source: [8])

After the group is created and PM selected, the GO waits for a period of time before tearing down the group and restarting the EMC protocol. This is done to ensure that the battery drain of the GO is minimal and that groups can be established with all the devices for the maximum period of time. Upon restart, the new GO is elected and the process is repeated.

In order to overcome the limitations of stock Android already discussed, the authors decided to modify Android's source code. By doing this, multi-group bidirectional unicast communication is allowed. Also, the issue of IP addresses assignment is mitigated by giving GOs different addresses.

Finally, in order to validate the protocol the authors created a chat application, which runs autonomously without any user interaction, apart from the messages to be sent. Manual override buttons are also present in the application for users to manually control the group creation and teardown.

K. Liu *et al.* propose a new implementation of MANET using Wi-Fi Direct in stock Android, in [9]. It is the authors' belief that MANETs using this technology can be used in LTE offloading systems. This implementation has the following properties:

- All devices must have the same setup, *i.e.* same functionalities.

- The devices must be ready to be discovered, connected and to transmit
- The MANET is dynamic so devices must be able to join different groups in the network
- All devices are able to leave or join the network, making the MANET not depend on any device

This solution follows a different approach than the previously presented. According to the authors, in order to achieve all of the properties listed above, all devices must become GOs when there is no data transmissions, creating a topology similar as the one in Figure 2.16.

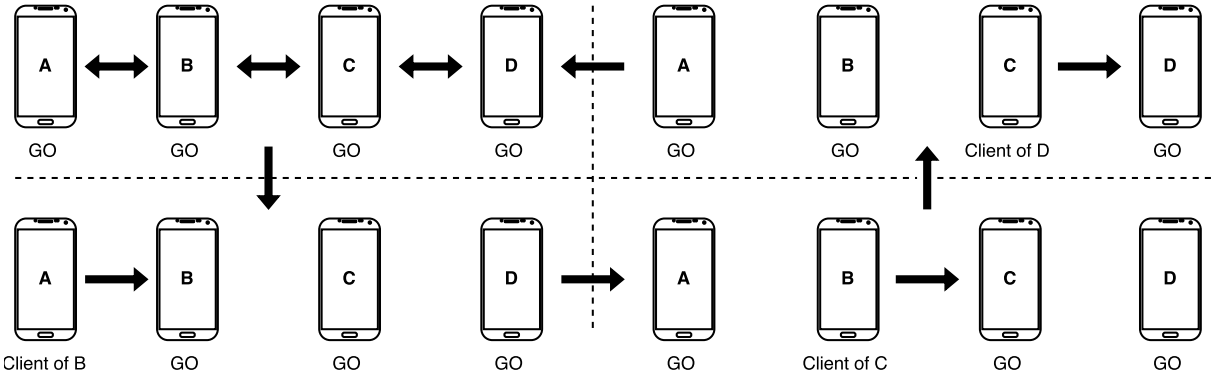


Figure 2.16: Wi-Fi Direct MANET topology (adapted from: [9])

The transmission cycle is as follows: the device with data to transmit must first remove its GO status and connect to the destination as GM, via the Wi-Fi Direct interface. Once the connection is established, the devices may communicate between them. After the transmission, the device acting as a GM disconnects from the group and becomes a GO again. This cycle is repeated whenever there is data to transmit.

This MANET topology is implemented in stock Android devices, as previously mentioned, and presents a distinct solution from the other works, since the devices in the network are constantly changing roles and groups inside the MANET. One disadvantage of this solution is that the status change of the devices are triggered by human interaction with the application, making the network somewhat dependent on human interaction.

2.3.3 Multi-Hop Routing

In the previous section some methods proposed by different authors were presented, in order to create a multi-group networks using Wi-Fi Direct. In this section the focus will be the routing algorithms implemented over some of these methods.

In [6], C. Casetti *et al.* propose a content-centric routing algorithm on top of the network topology proposed. Meaning each node knows what is the next hop to which it has to send the request for a specific content – see [16] for a detailed explanation on content-centric networks.

The authors introduce two data structures responsible for storing the information for content routing: Content Routing Tables (CRTs), providing the next hops to reach a certain content. These tables function similarly to standard IP routing tables. They store the MD5 hash of the IP address of the next hop.

There are three possible scenarios for filling the CRT:

- The simplest one where the content item is available within the group of the content requester, where the next hops of the all GMs is the IP address of the content provider.

- The second scenario is when the content is available in a different group, reachable through the group's relay node. This means the GO of the second group is connected as a legacy client to the first group and, according to the authors' scheme, all the GMs of the first group will have as next hop the group's relay node, except for the GO acting as a client. The next hop for the relay client is the IP address of the GO of the second group. Finally, the next hop of the latter is the IP address of the P2P interface of the content provider.
- The other scenario is when the content is available in a second group, reachable through the GO of the first group, which acts as a legacy client in the second group. Following the authors' scheme the next hop for all members of the first group is the IP address of the P2P interface of the GO acting as a legacy client. The GO of the first group will have as next hop the IP address of the second group's relay node. The relay client will then follow the steps from the first scenario.

The other data structure are the Pending Interest Tables (PITs), where the information about the destination of the content item is stored, they are the next hops from the reverse path of the content requests. When forwarding a content request, the nodes store the IP address of the node interface from where the request was received. With this mechanism, a "memory" of the content request path is created and utilized to then forward the content item to its requester. Upon receiving the content, the intermediate nodes forward the packet and remove the corresponding entry from the PIT. When multiple PIT entries requested the same content, the sending node replicates the packet and sends it to all the requesting devices, deleting all the corresponding entries. A content received by an intermediate node without any correspondent entry in the PIT is discarded.

The content registration, advertisement and request is done in two phases: the initial phase when a client advertises that new content is available, sending a message to the GO, which returns an Acknowledge Packet (ACK) as confirmation. The GO will then advertise within the group the availability of the new content, by sending a broadcast message to all GMs and waits confirmation from the relay node. This broadcast message is discarded at the IP layer by the legacy clients that are GOs of other groups. Thus, in order to create a multi-group advertisement, the relay client will send an advertisement to these clients and waits for the ACK.

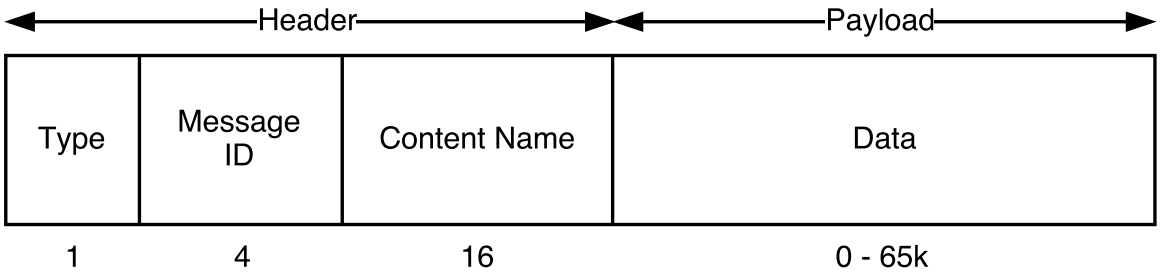


Figure 2.17: Application-layer message for content registration, advertisement, request and delivery. (adapted from: [6])

The message format can be seen in Figure 2.17, where "Type" refers to the purpose of the message, *i.e.*, Content registration, Content advertisement, Content data, Content request, Relay election or notification of GO role in another group and corresponding ACKs. "Message identifier" is used to identify what message is an ACK referring to. Content name is the MD5 hash of the content name. "Data" is the payload and can carry control or content data.

K. Liu *et. al* also provide a multi-hop routing implementation in [9]. The authors create a routing table composed by two sub-tables, one containing all peers that are directly discovered by the node and the

other composed by peers that are accessible via other peers, *i.e.*, multi-hop peers.

The first sub-table is created with a simple broadcast of the peer discovery signal and it will receive the responses from the devices in the vicinity, since in the authors' topology all devices are GOs if no data is being transmitted. The response messages should contain the MAC address of the destination devices, the gateway to reach the devices, in this case the node itself, since all nodes are neighbors and the number of hops to reach them, in this case zero.

After the first sub-table is complete all the nodes in the network have knowledge on who are their immediate peers. The nodes will then exchange routing table information between themselves to get a knowledge of all the multi-hop peers that may exist in the network. The second sub-table is filled proactively when nodes receive routing messages. The message-processing scheme is as follows: the device receives a message and checks its destination. If the destination MAC is not the device's MAC the device will look at its routing table and verify if an entry exists with that destination and where should the message be sent to, incrementing the number of hops, otherwise there is no route to the destination. If the destination MAC is the device's MAC, it will check the message type and infer if it is a routing message. If so, the device's routing table will be updated with the MAC address of the source of the message, the MAC address from where the node received the message, *i.e.*, the gateway, and the number of hops necessary to reach the source.

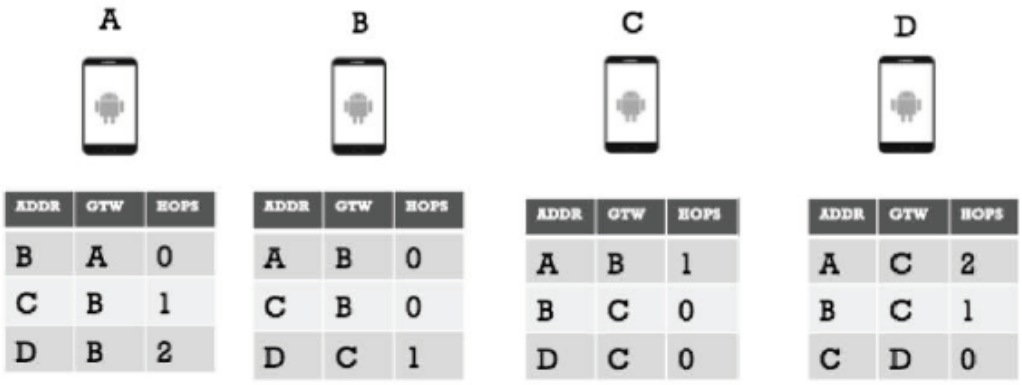


Figure 2.18: Wi-Fi P2P MANET routing table. (source: [9])

In Figure 2.18 it is possible to see a complete table for four different nodes. Each node is now capable of sending a packet to its destination, without having to use a broadcast mechanism.

Two different routing mechanisms have been introduced in this section. Each mechanism is supported by a specific network scheme. This is the main problem of routing in Wi-Fi Direct, since the routing algorithm is developed at the application layer thus being always dependent on implementation, making most of the algorithms not compatible among themselves. This problem also brings some limitations in the number of users that will integrate a specific network with a specific routing scheme.

2.4 Bluetooth *vs.* Wi-Fi Direct in Android Devices

In this section the data rates of both Bluetooth and Wi-Fi Direct will be analysed and compared. Since the data rates are technology specific, instead of conducting experiments that may not provide the most accurate results, a paper will be described, containing appropriate and accurate tests for these technologies' throughputs.

Bluetooth version 4.0 provides a theoretical, maximum throughput of *24Mbps*, from [17], whereas Wi-Fi Direct provides a maximum throughput of *250Mbps*, as stated in [18]. To provide a precise measurement of both Bluetooth and Wi-Fi Direct's actual throughput in Android devices, the work developed by A. Mtibaa *et al.* in [10] will be described.

The devices used were two Google Nexus 5 smartphones running Android version 4.4.4, with Bluetooth 4.0. These devices were tested in three distinct environments: indoor line-of-sight, in an underground walkway *350m* long and *10m* wide. Indoor with obstacles, consisting of a set of offices separated by walls, *4m* apart. Outdoor line-of-sight, in an outdoor walkway with *582m* long and *8m* wide.

Two sets of three experiments were conducted: one in each environment, using both communication technologies. In each experiment the throughput was measured by establishing a TCP connection beetween client and server and sending *5MB* files. The server then measured the total number of received bytes over *2s* intervals, computing the throughput in that period.

In Figure 2.19 it is possible to see the measured devices' throughput, using Bluetooth, in the three described environments, as a function of the distance between devices.

Figure 2.19a shows a plot of the throughput measurements in the indoor line-of-sight environment. It is possible to see that the maximum throughput achieved when both devices are in the same position, is *2Mbps*. The throughput drops to almost half its initial value at a distance of *20m*. At a distance of *140m* the devices are not able to communicate.

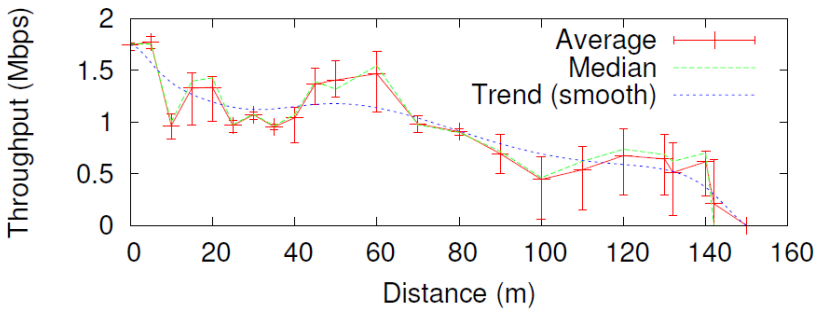
Figure 2.19b shows a plot of the throughput measurements in the indoor with obstacles environment. The maximum throughput is *0.9Mbps*, when the devices are in the same position. As the distance increases, the throughput decreases, in a linear fashion. At a distance of 5 offices (*20m*), the devices can no longer communicate.

Figure 2.19c shows a plot of the throughput measurements in the outdoor line-of-sight environment. A maximum throughput of *0.6Mbps* is achieved in the initial position. The throughput drops to, around, *0.2Mbps* at *40m*, slightly decreasing until a distance of *120m* separates the devices, preventing communication between them.

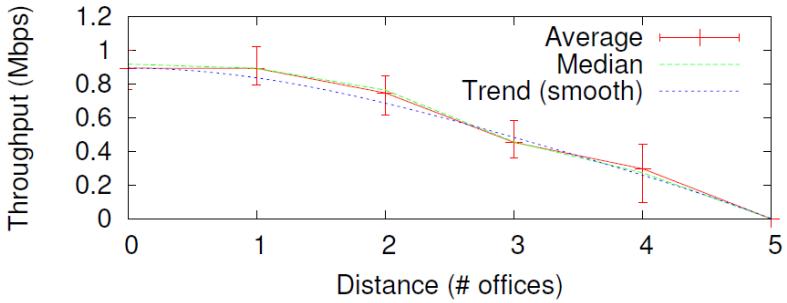
Overall, the devices using Bluetooth were able to communicate from *20m* with obstacles, similarly to what was observed in 4.1.1, to *150m* in the indoor line-of-sight environment. This disparity in the Bluetooth coverages comes from the phenomenon of signal reflections in the indoor environment, increasing the Bluetooth signal and consequently, its range. The throughput varied from *0.6Mbps*, in the outdoor line-of-sight environment, to *2.0Mbps*, in the indoor line-of-sight environment. This difference may be due to the channel interference, more prone to occur in the outdoor environment.

Comparing the obtained measurements with the theoretical values, it can be said that (1) the throughput was significantly lower than what was stipulated in [17] – although, TCP packets were used to measure the throughput, which may incur into further delays, as mentioned by the authors; (2) the communication ranges were much higher than the theoretical stipulation of *10m* for Class 2 devices, which most smartphones are – see [17] – however, for higher communication ranges, the throughput is much smaller, sometimes almost zero, as seen in Figure 2.19c.

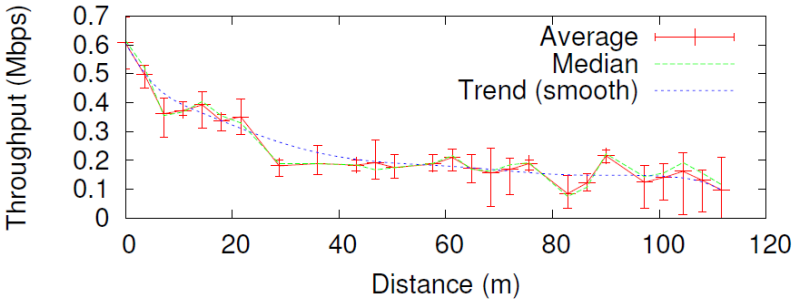
In Figure 2.20, it is possible to see the measured throughput of the devices, when communication via Wi-Fi. The authors state that, even though the showed results are obtained using Wi-Fi, the conclusions can be drawn to Wi-Fi Direct, as the results are redundant.



(a) Throughput in indoor line-of-sight.



(b) Throughput in indoor with obstacles.



(c) Throughput in outdoor line-of-sight.

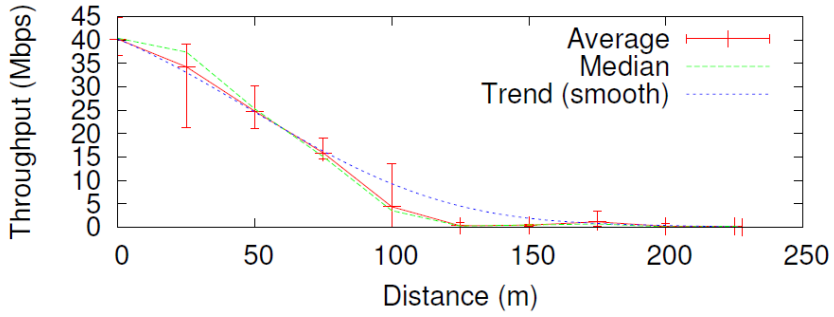
Figure 2.19: Android Bluetooth single-hop throughput measurements in different environments (source [10]).

Figure 2.20a represents a plot of the measured throughput in the indoor line-of-sight environment. At a distance of *0m*, the maximum throughput of *40Mbps* is achieved. It then linearly decreases until a distance of *125m*, where the throughput is almost null. At a distance of *200m* the throughput is, roughly, *1Kbps*.

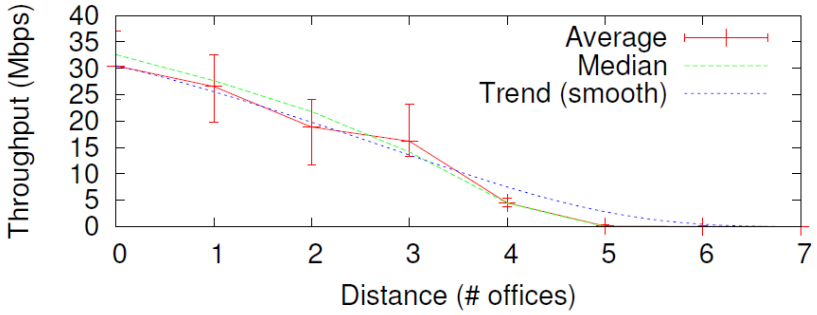
Figure 2.20b shows the measured throughput in the indoor with obstacles environment. At a distance of *0m*, the measured throughput is *30Mbps*. The indoor with obstacles environment presents a similar throughput decay as the one presented in Figure 2.20a. At a distance of 4 offices (*100m*), the throughput measured less than *5Mbps*.

In Figure 2.20c, the measured throughput in the outdoor line-of-sight environment is shown. At a distance of *0m*, the throughput measures *35Mbps*. However, at a distance of *50m*, it measured less than *5Mbps*. Despite what happened in the two previous environments, in the outdoor line-of-sight environment, the throughput decayed exponentially with the distance. At a distance of *200m*, the throughput reaches *0.8Mbps*, a much higher value than the one registered in the indoor line-of-sight

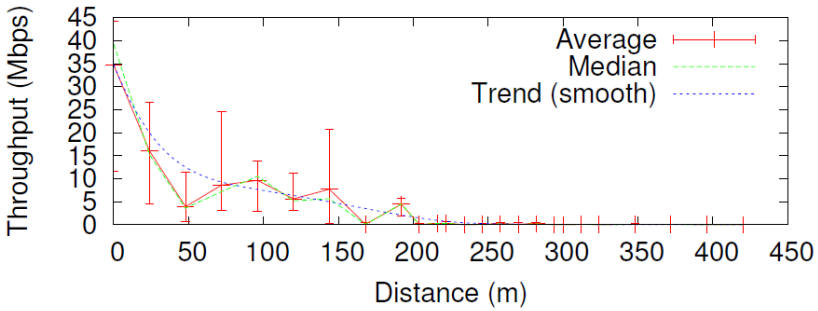
environment.



(a) Throughput in indoor line-of-sight.



(b) Throughput in indoor with obstacles.



(c) Throughput in outdoor line-of-sight.

Figure 2.20: Android Wi-Fi single-hop throughput measurements in different environments (source [10]).

After the brief analysis of the shown plots, it is possible to say that the indoor line-of-sight and indoor with obstacles environment present a similar throughput trend (in blue). In the outdoor line-of-sight environment, however, the throughput decays faster, in an exponential manner. This difference is due to the fact that propagation delay spreads are much smaller in indoor environments than outdoor environments. Having bigger delays, some reflections of the signal do not reach the server device during the *2s* measured intervals, reducing the throughput.

However, at distances longer than *200m*, outdoor environments show a much higher throughput than indoor environments. This is due to the fact that reflected and scattered signals are cancelled by signal attenuation, highly reducing the throughput and, as mentioned before, the reflection and scattering phenomena are more relevant in indoor environments.

It is also possible to state that, the throughput has never reached the theoretical value of *250Mbps*. The maximum throughput measured was in the indoor line-of-sight environment with a value of *40Mbps*, less than a fifth of the maximum theoretical value.

Further analysis of Bluetooth and Wi-Fi Direct's performance in Android devices will be shown in Section 4.1.

2.5 Ad hoc Networking Applications in Android

In this section some Android applications that make use of ad hoc networking will be presented, as well as the toolkits on which their development was based, if applicable. It is important in the scope of this work to know what are the offers on today's market on the ad hoc networking, to understand what has been done and what remains to be done and improved. These applications specifically make use of Wi-Fi Direct technology to establish communications or discover peers. It is worth mention that many other applications exist that create ad hoc networks but we are interested in the ones that use this technology in particular, since it will be the base to this work.

2.5.1 FireChat

FireChat is a cross-platform application that allows users to live chat with each other without access to the cellular or infrastructured network. It uses Bluetooth, Wi-Fi Direct or Apple Multipeer Connectivity to provide peer communication between devices.

FireChat is useful when there is a large concentration of users creating traffic that can, possibly, create some congestion in the infrastructured network. By using FireChat users are able to communicate between themselves with minimum delay, offloading the network and being independent on any service provider.

Users can create or join groups, *a.k.a.* chat rooms, to communicate with other users in the same group. Messages can be sent to multiple or single destinations, also messages can be either private or public: in the first case only the selected receiver/receivers will be able to see the message. In the second case the message is broadcasted to all GMs that form the chat room.

When a user sends a private message to another device, a private group is automatically created. However, senders might want to reach devices outside the groups where they are inserted, in that case the message is routed across the group as a private message to a GM that is connected to the infrastructure and can send the message to the destination, that will receive it upon establishing Internet connection.

FireChat also allows for other features, such as blocking the communication with specific users, sending of photos, following other users' FireChat activity, *etc.*.

This application is built with MeshKit, a Software Developer Kit (SDK) module with methods allowing for P2P mesh connectivity within devices using this module. It is built over Bluetooth, BLE, Wi-Fi Direct, ANT and other wireless protocols.

It works on three different mechanisms: cloud to mesh where devices connected to the Internet receive data from this link and share it with the other devices within its group, using one of the technologies mentioned above. The packets hop from device to device until it reaches the destination. This mechanism uses broadcast to diffuse the packets through the network.

The second mechanism is mesh to cloud where devices connected to the Internet forward data from devices without Internet connection but with a P2P connection established with the gateway devices, *i.e.* the ones connected to the Internet.

Finally, the peer-to-peer mechanism, where the Internet is not accessed and devices transmit data between themselves using only the P2P link. Data is routed within the groups to reach its destination. Multicast or unicast transmissions can be used.

2.5.2 Uepaa!

Uepaa! is one of the fastest growing P2P software companies, alongside with OpenGarden, responsible for FireChat. Uepaa! created Uepaa! Safety App, an application that aims to provide a security service for users who spend time outdoors and where cellular or Internet connection might not be available, such as mountaineers.

The app allows users to connect directly to the companies emergency call center even if no mobile network coverage is available in the area. This is achieved by forwarding the rescue message to users in the area using Uepaa!'s application. Similarly to FireChat's peer-to-cloud mechanism, in Ueppa! Safety App the message is routed through devices in the are until a device has a connection with either a mobile network or the Internet, and the rescue message, with the user's location is then sent to Uepaa!'s call center where the user can get assistance.

Uepaa! also developed the p2pkit a cross-platform SDK that allows for P2P transmission of beacons, with configurable information. P2pKit also focuses on the battery consumption, since it can be used in battery sensitive applications such as Uepaa! Safety App. It provides P2P communication with all the associated processes, such as discovery and formation of groups. Range estimation on who are the closest peers and how close are they is also included in p2pkit modules, which can be useful to efficiently manage the battery of the device by sending only the beacons to the closest peers.

P2pkit is the base for different applications. Nearby devices using p2pkit are notified when a proximity event takes place and, although the principle is the same, applications are tailored to respond differently to certain types of events, *e.g.* Uepaa! Safety App responds to the event of receiving a beacon by checking the device's Internet or mobile signal and tries to forward the message to the call center. other applications might trigger other responses, such as notifications to the current user, not forwarding the data but storing it and displaying it in the device.

2.5.3 Murmur

Murmur is an open-source, anonymous messaging application. Users can communicate with one another without Internet or mobile connection, similarly to the applications described above. However, Murmur uses Wi-Fi Direct and Bluetooth technologies to establish communications between the devices, creating a Delay Tolerant Network (DTN) in which the P2P transmission of messages occurs.

Much like every other P2P network, the more users Murmur has, the faster transmission of data is. Messages are broadcasted over the network and if no device is in the vicinities of the sender, the message is queued for posterior transmission, thus the delay tolerant network designation. Murmur allows users to post messages to the network, a system to upvote, store, share and search for posts.

The discovery and connection between devices is done using both Wi-Fi Direct and Bluetooth: each user advertises it is running Murmur via the Wi-Fi Direct interface with a specific name, "MURMUR:Bluetooth MAC", constantly searching for peers using a similar device name, *i.e.*, starting by "MURMUR". If a discovery is made, the application filters the Bluetooth MAC address from the device's name and creates a Bluetooth connection, exchanging the message. The actual exchange starts with a handshake where each device sends its contact list and the amount of shared contacts between the two is calculated. Secondly, the number of expected messages to be exchanged is sent and, upon agreement, each device transmits until that number is reached. Finally, when the exchange is complete, the received messages are saved to a local database and the devices set a backoff time before establishing other connections, avoiding redundant exchanges and unnecessary battery consumption.

Chapter 3

Prototype Peer-to-Peer Web Access over Bluetooth

In this chapter the developed peer-to-peer communication prototype will be analysed. This explanation will address the decisions about the technology and protocols to use as well as the architecture and workflow of the application.

It will begin with a brief summary of the steps taken to reach the developed application's architecture and functionality. After the overall application's objectives and high-level architecture is described, a section on the lower-level architecture and processes of the application will be presented. Finally, there will be a section describing the conclusions on the developed work, from its limitations to the possible future work.

3.1 Design Choices

This section will provide the explanation and justification of the choices made for the application type, ad hoc communication technology and ad hoc routing protocol.

3.1.1 Application Type

The choice of the target application was mainly based on the innovation criterium. For this purpose, an investigation was made on already existing peer-to-peer applications, see Section 2.5. Most of the existing applications are designed to support text messaging with much of the developed work focusing on recreating popular applications, such as Messenger and WhatsApp, using a peer-to-peer network to route the messages to their destinations. Beaconing and geolocation messages are also not completely innovative, since applications such as Uepaa!, see Subsection 2.5.2, already implement peer-to-peer applications where these types of messages are exchanged. Given this reasoning it was considered that the most innovative application type would be web browsing, supporting a multi-hop hot spot in an ad hoc network.

3.1.2 Ad Hoc Communication Technology

The choice of ad hoc network technology was another important step. The first and most obvious choice would be to use Wi-Fi Direct in both advertising and communication between devices, since this technology offers the best features to transfer files around *1KB*, in both range and data rate of

transmission. However, during the development it proved to be impossible to continue with this approach, as Wi-Fi Direct's current Android implementation does not allow for devices to transfer files without active user participation, also mentioned in [5].

Given this drawback a shift to Bluetooth was made, and a hybrid version of the application was created, where the advertisement would be done via Wi-Fi Direct and the actual transfer of the web pages would be made via Bluetooth. This method also proved to be infeasible, due to security issues, since the the devices would have to display their Bluetooth MAC addresses in their Bluetooth name, making them much more vulnerable to possible attacks.

The advertisement was also a topic of debate. Two possibilities were presented: to use simple Bluetooth connections to advertise to each peer or to use BLE beacons as an advertisement method. The first method is slower, since the advertising device needs to create a connection with each discovered peer, in order to transmit its advertisement. The second method would offer a better advertisement mechanism, since it is faster and consumes less resources than the traditional Bluetooth connections (see Subsection 2.1.3), but it gives every device the ability to capture the advertisement, bringing some security issues.

The beacon advertisement is a tempting mechanism to implement, given all its advantages. It comprises two components: transmission and ranging. Transmission occurs when a device wants to issue an advertisement to a certain region[1]. Ranging is the act of listening to beacon transmissions in a certain region. During the implementation of these two components it was possible to verify that the current Android implementations – tested with Android versions 6.0 and 7.0 – are not able to reproduce BLE's transmission mechanism, being only able to use the ranging.

Given these facts, the application was developed using simple Bluetooth connections for both advertisement of the devices and data transfer. Although this is not the best technology for communication in an ad hoc network, it is the only one that currently meets all the requirements for this work. In Section 5.2, a brief discussion on the changes that need to be made to Wi-Fi Direct's implementation in Android will be presented, and why developers could benefit from these changes.

3.1.3 Ad Hoc Routing Protocol

The last step was to create/modify a routing algorithm to control the destination of each incoming message. The requirements are simple: the algorithm simply needs to save the next hop of the shortest path leading to a device with Internet connection.

Destination-Sequenced Distance Vector (DSDV) is a routing protocol used in ad hoc mobile networks. This protocol uses a routing table where it stores in each entry the destination, next hop, number of hops to reach the destination and a sequence number to avoid routing loops. The devices exchange full routing tables, creating a network where every device has total knowledge of the topology. However, DSDV requires the exchange of routing tables and their regular updates, wasting bandwidth – see [19] for more information on DSDV.

AODV is also a known routing scheme used in ad hoc mobile networks. It establishes paths in a reactive way, *i.e.* only when a device wants to retrieve a web page does this protocol search for a path in which to send the packet – see [20] for the full specification of this scheme. This scheme is not the most reliable, since the requesting device should know beforehand if it is capable of reaching the Internet, otherwise users will uselessly experience long periods of path requesting and advertising every time they request a web page.

The proposed application does not have a specific destination as a goal, *i.e.*, a device A does not want to transmit to B, it only wants to reach a node with network access with the minimum amount of

[1] A beacon region is the physical space reached by the advertisement.

hops. Hence, the sending device will have no need to know the elements of the routing path, other than the next hop node. Thus, in order to accommodate such requirements some modifications were made to DSDV. Firstly, to reduce the signalling overhead used by DSDV, the developed version will simply exchange the best next hop estimate of the current device. Secondly, instead of a periodic exchange of tables, devices will update their tables every time a new Advertisement message is received and they only exchange routing information when a new best path is received.

After a flexible period of time, each device erases its routing table and proceeds to restart the discovery and advertisement protocol – see Subsection 3.2.2. This allows devices to adjust their routing tables to the network's changes, such as erasing routes that no longer are available and replacing them with newly created ones. The period is flexible and can be changed according the dynamics of the network.

The proposed protocol provides a loop-free path for every request, since every node only advertises the best hop distance estimate. The downfall is that the requests' paths are not flexible, *i.e.*, a Request message is always sent through the best path and, if a link along that path is broken, the Request message is discarded. New paths are only established periodically, when a new round of advertisements is performed, as mentioned in the previous paragraph.

A different routing protocol could be deployed, one that has more flexibility, regarding the requests' paths. A possible approach could be to use a similar Advertisement message structure as the one used in DSDV. The MAC address of the advertisement creator would be added, as well as a sequence number. These new message fields would be used to assess if the received advertisement is newer than the previously received advertisement from that node, or was simply delayed. With this mechanism, it would be possible to establish several secondary path if the best path was inaccessible.

However, to establish such protocol, would add complexity and consume time resources during both the advertisement and web page exchange processes. Nevertheless, this option could be implemented, in an application for which reliability is more important than delay.

Since this work aims to to develop a peer-to-peer web access, the time of data exchange is crucial, as this application would be useless if the time to access the Internet is too big. Thus, the first, lighter routing protocol was implemented.

3.2 Architecture

In this section the architectures of both framework and application are presented.
The developed framework and application's main structure are similar and consist of:

- A routing algorithm that controls the destinations of the packets in each hop.
- A communication technology, *e.g.* Bluetooth or Wi-Fi Direct, handler, for both discovering nearby devices and to establish communication sockets with peers.
- A set of functions capable of analysing incoming messages and deciding which is the next step to perform, in order to complete the requests/advertises.
- Finally, in the application, a user interface is necessary, where the incoming messages can be seen and analysed (for debug purposes), also providing a text area for the user to enter the requested data.

In Figure 3.1 an overview of the developed application is shown. It can be seen that there are two main parts of the application: the main activity and the *BluetoothService*.

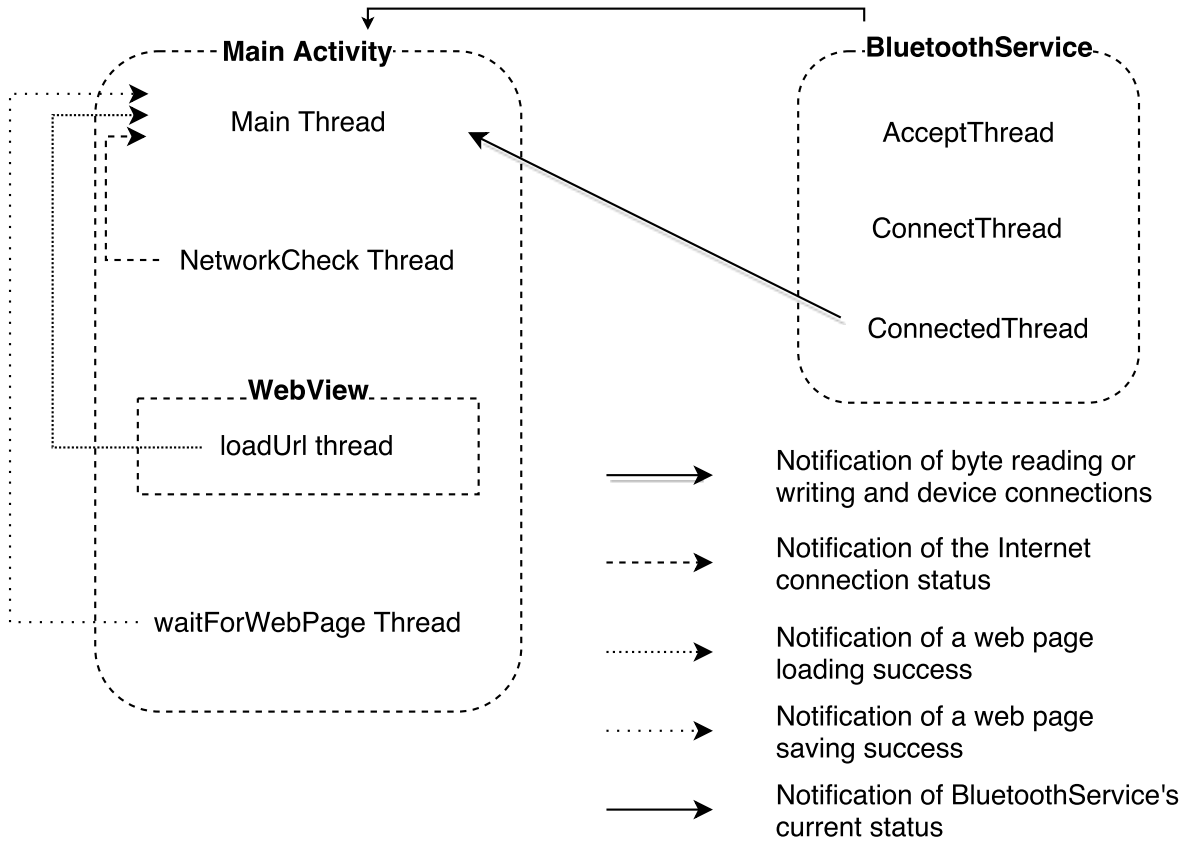


Figure 3.1: Overview of the different parts of the application and the communication between the threads running in each part.

- The main activity is where most of the processing logic is executed, from the analysis of the received data to the management of the routing tables. It comprises four threads: the *Main* thread seen by the user, also known as the user interface thread; the *NetworkCheck* thread, used to perform the query regarding the Internet connection status of the device; the *LoadUrl* thread, created when a *WebView* object is used to show or download a web page; finally, the *WaitForWebPage* Thread used to ensure the requested web page is successfully saved in the device.
- The *BluetoothService* is used to manage the Bluetooth connections between devices, it comprises three threads: the *AcceptThread*, the *ConnectThread* and the *ConnectedThread*. Each of these threads will be explained in detail in the next subsection.

Inside the main activity the different threads communicate with each other, mainly to notify the *Main* thread of a certain occurrence that may lead to a new set of actions. However, communication is also done between the main activity and the service, usually when the former needs to access the *BluetoothService*'s current status.

3.2.1 Bluetooth Connections

In this subsection, the Bluetooth connections between devices and their possible stages will be described and analysed.

Since the communication technology is Bluetooth, the application must have a thread[2] that handles

[2]Thread is a sequence of instructions managed independently, usually by a scheduler from the operating system. See [21] for more documentation on threads.

the listening and acceptance of connections and the data transfer between devices. This corresponds to the *BluetoothService.java* class. It creates and listens to insecure communication sockets, allowing devices to exchange data without user intervention.

The service has three different threads within it: a thread for the acceptance of incoming communications – *AcceptThread*; a thread for the initial exchange of vital information before the connection – *ConnectThread*; and a thread for the actual exchange of data between devices – *ConnectedThread*. The connection of devices goes through each of these stages, allowing the *Main* thread to get information on the connection status of the device at a given time.

There were two different approaches to the connections of devices: either the devices stayed connected as long as possible until a new connection was requested, or the devices stayed connected for the minimal amount of time for data to be transferred. The second approach was used, since it is the one that is more energy efficient, draining a lower amount of energy from the device batteries.

Bluetooth communication between devices is only allowed if both devices have matching ”credentials”, providing a minimum amount of security to prevent against possible attacks. An application specific name and UUID represent the application, making it distinguishable from any other, since the UUID of an application should not be repeated in any other.

To notify the *Main* thread about what is being done by the *BluetoothService*, a handler is created, acting as bridge between the two parts. It can be used for various actions, such as: retrieving the status of a connection, assessing if a device is ready to receive a file, *etc.*

Bluetooth connections can be in one of four stages: listening, connecting, connected and not enabled. Each thread mentioned earlier is the implementation of a stage.

In the next subsections each stage will be explained in detail, providing a better understanding on how the connections are managed.

3.2.1.1 Listening

A device is said to be in the listening stage when it has no ongoing connection but it is waiting for a new one initiated by another device. It is the ”first” stage of the *BluetoothService*, since every time the service is initiated, the device is moved to this stage.

In Figure 3.2 the flow diagram of the performed actions from the point of view of a device that is waiting for incoming connections is presented.

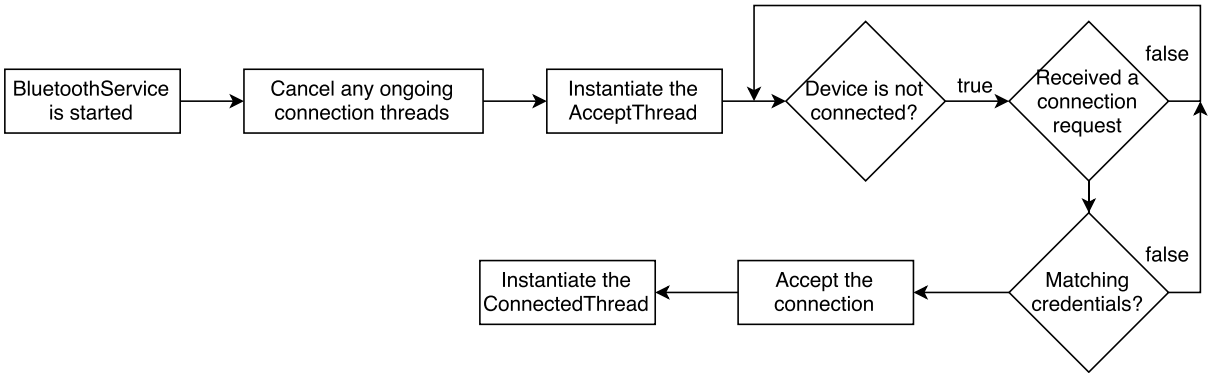


Figure 3.2: Flow diagram of the performed actions of a device listening for incoming connection requests.

As was said previously, each connection stage is associated with a thread class. The listening stage is associated with the *AcceptThread*, which is instantiated every time the device enters this stage. The framework's connections have short durations and each time a new connection is performed, the service

is restarted, entering the listening stage. Hence, it is imperative to assure there are no memory leaks or communication sockets left open. Thus, before the listening thread is instantiated, it is verified if any other Bluetooth connection threads are running, in which case they are shut down.

Once all the requirements for the instantiation of a new thread are met, a communication socket is created in which the device listens to incoming connection requests. To implement the listening mechanism, a loop is created whose purpose is to block the thread until a new connection is received, an error occurs or the user shuts down the application.

Whenever the device receives a new connection, it verifies the ”credentials” sent in that connection request and compares them with the ones defined in its *BluetoothService*. If they match, the connection is accepted.

Upon successfully accepting a connection, the method assesses the status of the newly formed connection and decides upon it. If everything goes as planned, the connection should now be in the connected stage. Once this migration between stages is concluded the instance of the listening thread is destroyed and a new connected thread is created.

3.2.1.2 Connecting

For a Bluetooth connection to be performed, at least two devices must exist, one of them acting as the connection initiator. In Figure 3.3 the actions performed by a Bluetooth connection initiator during the connection establishment period are presented.

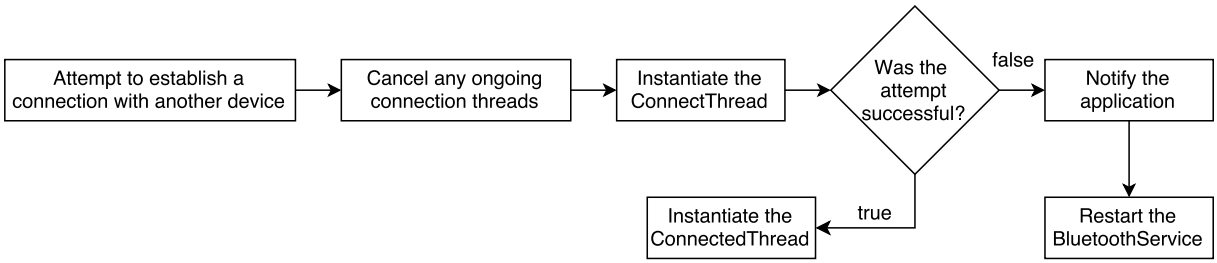


Figure 3.3: Flow diagram of a Bluetooth connection initiator’s performed actions.

Similar to the listening thread initiation, the *ConnectThread* is only instantiated after all the previous Bluetooth connection threads running in the device are shut down. Also, to avoid delays during the connection process, the Bluetooth discovery process is shut down. Only then can the initiator attempt to establish a connection with the desired device.

To identify the receiver of the connection request, the device needs to retrieve the MAC address of its counterpart – this will be explained in Subsection 3.2.2. Once this is attained, the connection initiator can attempt to create a communication socket between both devices, using the previously defined Bluetooth ”credentials” and the receiver’s MAC address. If the MAC address is valid and corresponds to a device within reach, the connection is requested successfully.

In case of failure due to the rejection of the connection from the receiver, a notification is sent to the *Main* thread, notifying the connection was not successful and the service is restarted.

However, if the sent connection request is accepted by the receiver, the connection state is assessed and, if no interruptions or errors occur, the connection stage is moved to the connected stage, indicating both devices are able to send and receive bytes from their counterpart.

3.2.1.3 Connected

When the connection reaches the connected state, both the connection initiator and the receiver devices are linked through a Bluetooth communication socket. To start the receiving and transmitting mechanisms, a *ConnectedThread* is created, after all ongoing threads are cancelled to avoid conflicts between communications or memory leaks. To notify the *Main* thread about the connection success, the service sends the counterpart's Bluetooth identifier, *i.e.* its Bluetooth name, to the *Main* thread.

To understand how data is transferred between the two connected devices it is necessary to explain two different mechanisms: the writing of data and the reading of data. By creating a socket linking both devices, two data streams are also created implicitly, where one provides a stream to write data into the socket and the other provides a stream to read data from the socket – see Figure 3.4. Both writing and reading data is in the form of byte arrays that can be afterwards manipulated into the desired format.

Figure 3.4: Communication channel with input and output streams.

Writing bytes into the socket is done through an output stream, name given to the part of the stream that handles the writing. For any data format, the principle is the same: to convert the data into a byte array that will be sent through the stream. It is important to note that the channel has a limited size buffer shrinking the arrays' size, thus if the data to be exchanged has a large amount of bytes, it is wise to perform segmentation. The writing process is asynchronous, since it is only executed at a certain time following a certain trigger, *e.g.* user input. It is possible to notify the *Main* thread about the sent bytes in case they are relevant to other processes, *e.g.* notifying that a text message was sent and prepare to receive/send an image.

Reading data from the socket is done through an input stream. Since the output stream only writes byte arrays into the socket, the input stream will only read byte arrays. This property burdens the developer with the task of parsing the received bytes, *i.e.* identifying them and then converting them into their original data type. Also, as mentioned before, the communication socket has a limited size buffer, thus it is also the developer's responsibility to perform segmentation and reassembly of the sent data.

In Figures 3.5 and 3.6, the actions performed to read and write data from and to a Bluetooth communication socket are shown, respectively.

To send and receive different data formats within the same application, it is needed to develop a protocol to support the required multiplexing functions. Further along this chapter it will be explained how this was done in the developed application to exchange both text messages and web pages.

3.2.2 Discovery and Advertisement Protocol

In order to reach an hotspot, a device must first fill its routing table. This process is done by a series of neighbor discoveries[3] and advertisements[4], explained in Subsection 3.2.2.1 and 3.2.2.2, respectively. Each device will advertise its best estimate to reach the Internet. This advertisement will be processed by peer devices, helping them to fill their own routing tables with the best possible routes.

[3]Discovery is the act of finding nearby devices with Bluetooth on.

[4]Advertisement is the act of notifying peer devices of the cost of relaying a Request message to the device issuing the advertisement.

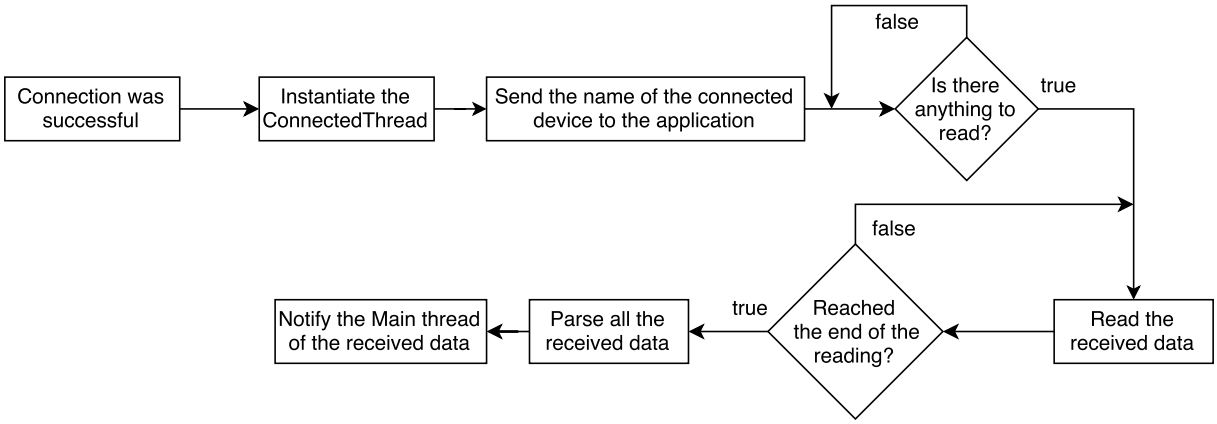


Figure 3.5: Flow diagram of the performed actions of a device reading from a Bluetooth communication socket.

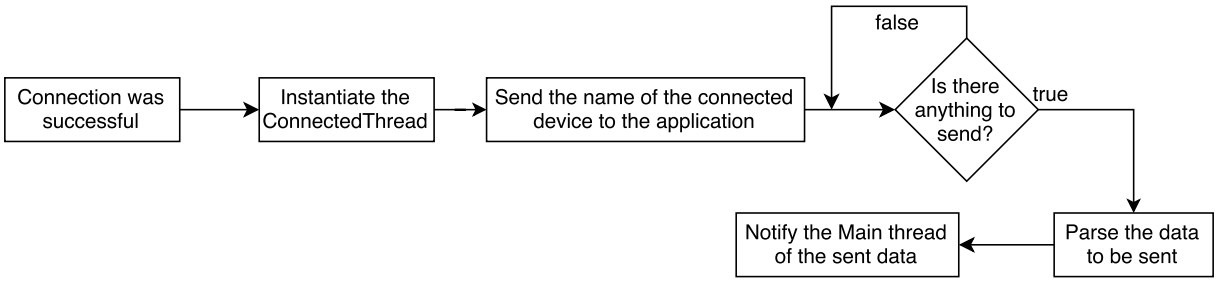


Figure 3.6: Flow diagram of the performed actions of a device writing to a Bluetooth communication socket.

In Figure 3.7 the format of an Advertisement message is shown. The type is a feature common to all messages exchanged in this application. It servers the purpose of identifying which type of message is being received. It can take four different values: *ADV*, *RQT*, *RSP* and *FAIL* for an Advertisement, Request, Response and Fail message, respectively. In this case it will take the value *ADV*. The Bluetooth MAC identifier is always the MAC address of the Bluetooth adapter of the sender. It is then used by the receiver to populate its routing table. Finally, the estimated number of hops is the lower number of hops that separate the sender from the Internet Access Point plus one hop, corresponding to the connection between sender and receiver.

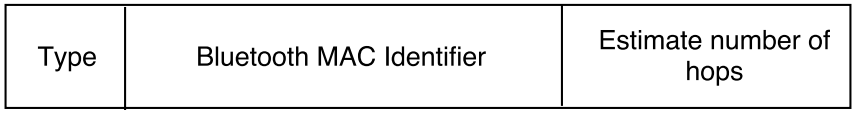


Figure 3.7: Advertising message format.

Each device has two different routing tables: one that is used to store the information retrieved from the advertisement process, providing the best path for routing the Request messages to the Internet Access Point; another used to route a Response message back to the original sender of the Request message that originated the Response, as explained in the next subsection. For simplicity, the first routing table will be referred as Routing Table, while the second will be referred to as Response Table, since it handles the routing of Response messages.

In Table 3.1, an example of a Routing Table is presented, where the first entry is always populated

Next hop node (MAC address)	Number of hops
Own MAC address	0 or 16
Device A's MAC address	Estimate through A
Device B's MAC address	Estimate through B
...	...
Device Z's MAC address	Estimate through Z

Table 3.1: Routing Table example and format

with the device's own MAC address and its hop distance estimate, which is either 0 or 16[5], meaning the device has an Internet connection or not, respectively. Every time a device receives an Advertisement message, it populates this table, either by adding a new row or updating an existing one, with the information contained in the message – see Figure 3.7. When the routing table is done being populated, each node knows its immediate peers and the best estimate through each one of those peers, giving it a full knowledge of its vicinity, *i.e.*, the sub-network created by the peers that are reached by the device using a single hop.

It is important to start by understanding how the routing tables are created and, then, how and when they are populated. Both tables follow the same principle: they are objects that relate keys to values. To a single key corresponds a single value, thus not creating duplicate entries.

In the scope of this subsection only the Routing Table will be explained – see Subsection 3.2.3 for the description of the Response Table. The keys of this table correspond to the MAC address of the next hop while the values correspond to the respective estimated number of hops. When querying the table for a specific MAC address it should return one and only one estimate for the number of hops.

There are three important things that the application needs from this table (1) to get the absolute minimum hop distance estimate to the Internet Access Point, in order to retrieve the device's best path; (2) to get the corresponding key to the minimum hop distance estimate to the Internet Access Point, in order to retrieve the receiver to whom this device should send the message; (3) to update or add a row with a new key-value pair, in order to add newly received advertisement information.

To be able to get the absolute minimum of the estimated number of hops, it is necessary to compare all the existing values of the table. Since it only contains the information of its immediate peers, this process is fast and does not interfere with the rest of the application.

Having the minimum hop distance estimate to the Internet Access Point from the Routing Table, the MAC address associated with that entry is retrieved. This MAC address belongs to the device providing the best next hop to reach the Internet Access Point.

Finally, the addition and update of Routing Table rows is done whenever an Advertisement message is received. The advertisement provides the MAC address and estimated number of hops. Having this information, it is possible to insert the retrieved key-value pair into the existing table. As mentioned before, one key maps to a single value, thus in case this key already maps to a value, the latter will be overwritten with the new one. In case the key does not exist, a new row is created mapping the key

[5] 16 was chosen to be the representation of infinity or inability to reach a destination, since it has been represented by this number in various protocols, for instance in Routing Information Protocol (RIP), see [22]

to its specific estimated number of hops. Note that, in case of a table update, the application does not compare values, *i.e.*, it does not check if the new estimated number of hops is better than the previous one, since the previous route may have been broken and thus the sender of the Advertisement message always advertises the most recent one.

Since the maximum number of hops to reach an Internet Access Point is capped at 16, the problem of a request being in an infinite loop does not apply. However, if a route link is broken, *i.e.*, a device in the routing path is disconnected or moves out of range, the request will fail until a better estimated number of hops is advertised – see Subsection 3.2.3.1, for the failing notification process. To avoid the permanent failure of a request sending, the discovery and advertisement process is repeated in a certain amount of period (default time period is 5 minutes), refreshing the Routing Table.

3.2.2.1 Discovering Peers

Now that the Routing Table access and modification mechanism is explained, it is possible to discuss where and when it is useful. However, before that, there are still some indispensable features that need mentioning:

- During the peer discovery process, several peers may be found, making it necessary to create a place to store them, for further examination. To achieve this a list of neighbor Bluetooth devices is created, where each device corresponds to a found peer in the discovery process. This list is then used to retrieve information from these devices, in order to successfully establish communication sockets with them.
- To prevent the discovery process of dragging for longer than expected, using processing resources and damaging the performance of the rest of the application, it is necessary to identify if this process is finished and notify the *Main* thread.

Figure 3.8 depicts the sequence of actions that the application executes to successfully perform the discovery process.

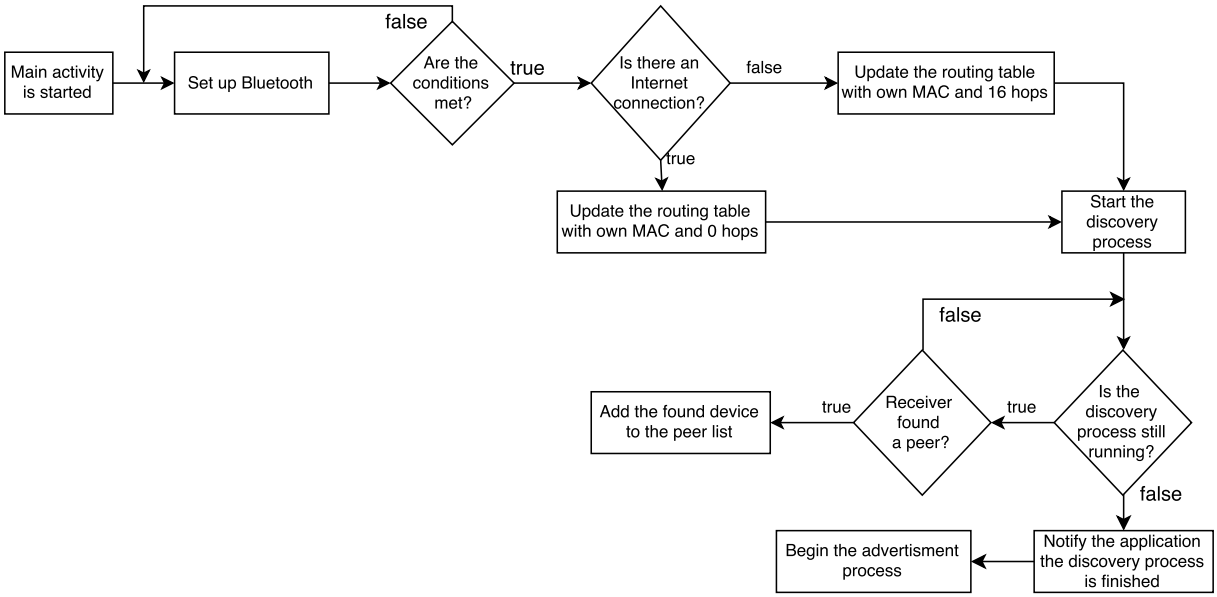


Figure 3.8: Flow diagram of the discovery process.

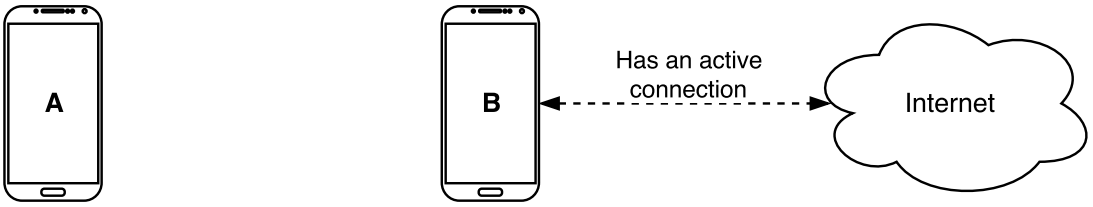
Before beginning the discovery process, the application needs to ensure that all the conditions are met. The Bluetooth status is the first to be checked. The device should have Bluetooth enabled and

it should be discoverable for an unlimited amount of time. If the Bluetooth is not enabled, the device cannot begin the discovery process and if the device is not discoverable its peers are not able to send their advertisement to it.

Once the Bluetooth is set up, the initial update of the Routing Table is performed. A quick network check indicates if the device has an active Internet connection. If this is the case, it will add a new row to the routing with its own MAC address and 0 hops. Otherwise, it means that the device is currently unable to establish an Internet connection. The same process is performed but, instead of 0 hops, the hop distance estimate will be of 16 hops, for reasons already explained.

The device is now finished with the first step of advertisement, filling the Routing Table with its own MAC address and estimate. It is now possible to start the discovery process and advertising this same hop distance estimate to the discovered peers.

Figure 3.9 demonstrates two devices, one without an Internet connection (left) and one with an active Internet connection (right). At this point both devices should have exactly one entry at their Routing Tables, since they have not communicated with any other device, but they have established their position in the network, *i.e.*, whether they have an Internet connection. Device A is unable to reach the Internet, so it adds to the Routing Table an entry with its own MAC address and a number of hops estimate of 16. Device B, on the other hand, is able to directly reach the Internet, thus having an entry with its own MAC address and a number of hops estimate of 0.



Next Hop's MAC address	Number of Hops
Own MAC address	16

Next Hop's MAC address	Number of Hops
Own MAC address	0

Figure 3.9: Example 1: State of the two devices after the initial step of the advertisement process is over.

The management of the discovery process is relatively simple: the receiver needs to be able to assess the discovery process status, *i.e.*, if it is starting, running or finished. The management of peer discovery is more complex: the receiver needs to retrieve each peer's information and store it in the peer list previously mentioned. A new entry is added only if an entry for the same peer is not already included in the table. Once the discovery is finished, the receiver needs to be able to notify the rest of the application that it can proceed with the advertisement process.

3.2.2.2 Sending an Advertisement Message

The discovered peers are now stored and the device can start the advertisement process. The peer list is iterated through and each item is analysed to understand if it should receive an Advertisement message or not. The analysis consists of checking if the peer fits in the desired category: in this case a smart phone using the developed application.

In Figure 3.10 the flow diagram of the advertisement process is presented.

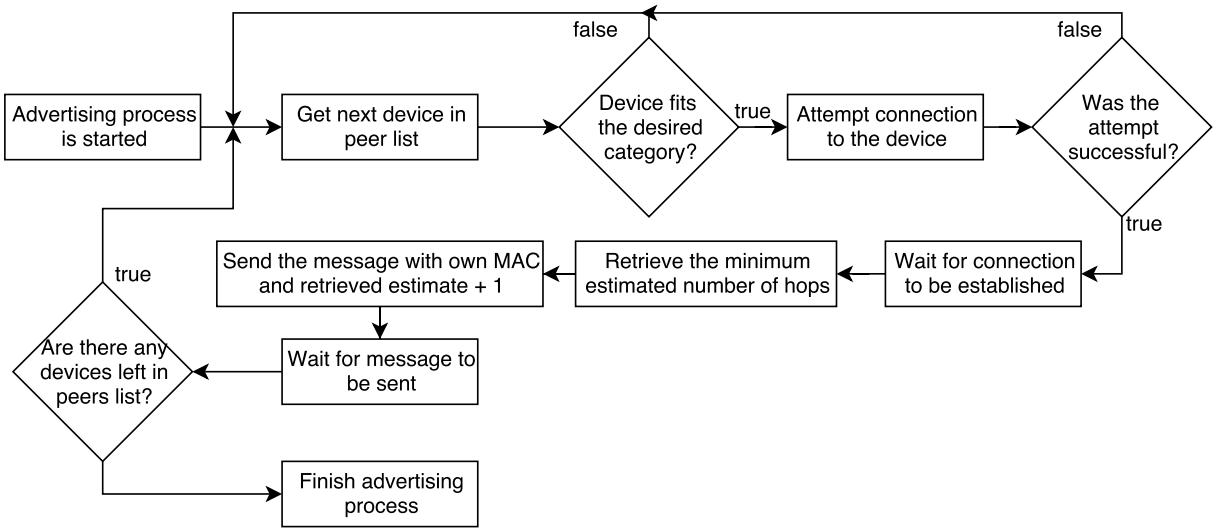


Figure 3.10: Flow diagram of the advertisement process from the point of a sender.

Since the number of Bluetooth devices is growing, it is plausible that the discovery process found a peer that does not fit into the desired category, such as a sensor or an headset. To avoid this, a verification is performed to assess if the peer is a smart phone. Overlooking this may be costly in terms of time and computing as the number of attempted connections will increase.

Once the necessary checks are performed and the peer device is eligible to receive an Advertisement message, the device attempts to establish a Bluetooth connection, using the mechanism described in 3.2.1. Before sending the Advertisement message, the application needs to ensure the connection has been successfully established and both devices are ready to write and receive bytes, through the respective streams.

If the conditions to send the message are met, the device queries its Routing Table for the minimum estimated number of hops, explained in Subsection 3.2.2. This value is incremented and added to the avertisement message. The Advertisement message is then sent, following the format seen in Figure 3.7, where the Bluetooth MAC identifier is the device's own MAC address. This process is then repeated for each peer device found.

3.2.2.3 Receiving an Advertisement Message

In Figure 3.11 a simplified flow diagram of the actions taken by the receiver of an Advertisement message is shown.

On the advertisement receiver device, the *BluetoothService* notifies the main activity of the message reception and, as mentioned before, this is achieved via a handler, used to pass the received message from the *BluetoothService* to the main activity.

Once the main activity is notified of the message reception, it analyses the context of the application, *i.e.*, which data format is the application expecting to receive at that specific point in time. Since this subsection's scope is the advertisement process, only text messages are expected to be sent and received.

The connection to the advertiser is no longer required, thus the *BluetoothService* is restarted in order to allow the device to start listening to incoming connections. Simultaneously, the received text message is submitted to analysis, in order to identify what type of message has the device received, from the different possibilities: Advertisement, Request, Response or Fail message.

By analysing the type field present in the message, see Figure 3.7, the message is identified as

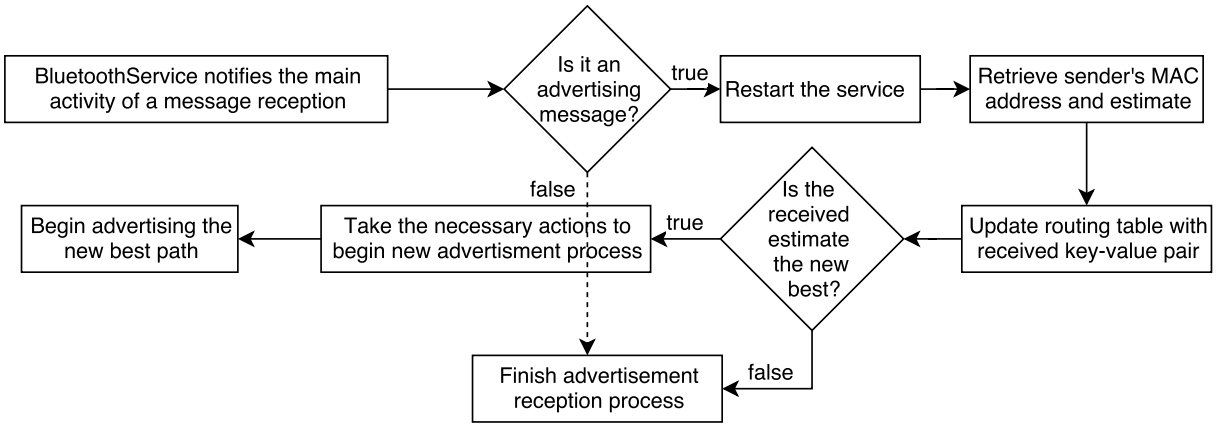


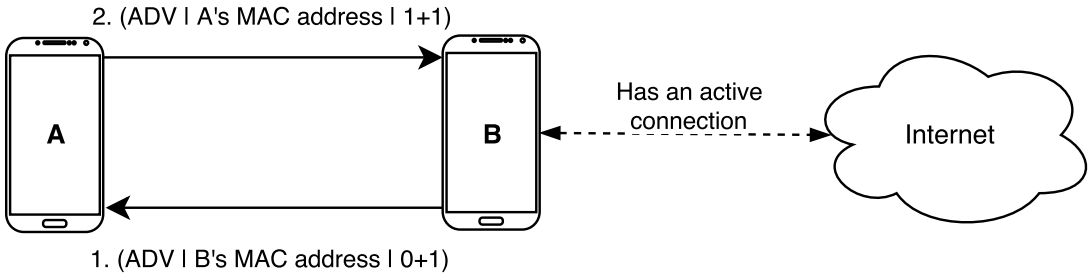
Figure 3.11: Flow diagram of the advertisement process from the point of a receiver.

an Advertisement message and submitted to the chain of actions specific to that message type. The first step is to extract the information contained in the message, *i.e.*, the sender's MAC address and the estimated number of hops to reach the Internet.

Having these values, the device compares the received hop distance estimate with the best from the ones stored in the Routing Table, process described in Subsection 3.2.2. If the comparison concludes that the received hop distance estimate is not smaller than the current best, a new entry is added to the Routing Table is added with the received hop distance estimate and MAC address, since it provides a possible alternative route if the current best path to reach the Internet Access Point is broken.

Otherwise, if the received hop distance estimate provides a better path to reach the Internet Access Point, the Routing Table is updated, the device then takes the necessary steps to start a new advertisement process, this time advertising the new best hop distance estimate, previously described.

In Figure 3.12, the example from Subsection 3.2.2.1 is extended.



Next Hop's MAC address	Number of Hops
Own MAC address	16
B's MAC address	1

Next Hop's MAC address	Number of Hops
Own MAC address	0
A's MAC address	2

Figure 3.12: Example 1: State of the two devices after the advertisement process is concluded.

Now, device B has advertised to its peers, which include device A. Upon receiving this message, device A proceeds to store the information in its Routing Table, followed by an advertisement from itself, since the new hop distance estimate is better than the previous one. After this process is concluded, both devices have the Routing Tables displayed in Figure 3.12.

If A advertises first, the result will be the same, since when B is advertising, A will still receive a better

hop distance estimate and advertise again. When B receives the second Advertisement message from A it will overwrite the previous entry, maintaining the same values.

After the discovery and advertisement phases, each device has a full view of its vicinity and it is possible to establish a network for Internet access using the best possible path. The next subsection will refer to the next step, where the devices already have their Routing Tables populated and are ready to exchange web pages.

3.2.3 Web Page Exchange Protocol

This subsection will cover the developed web page exchange protocol. To manage the exchange of the web pages, two messages are exchanged between devices (1) a Request, containing the web page's Uniform Resource Locator (URL), which is sent from the request initiator to the Internet Access Point; (2) a Response message, managing the return of the web page, guaranteeing it follows the correct path to its destination. The Response message will follow the Request's inverse path, starting in the Internet Access Point and ending in the request initiator.

To aid devices sending Response messages in the correct path, a Response Table is created. Once a device has received a Request message and it has an Internet connection, it simply sends back the Response message from where it received the Request. But, in the case this device is a "bridge" node, *i.e.*, a node that forwarded a Request message, with no direct Internet connection, it may have received requests from different devices and it still needs to be able to differentiate each message and decide which device to send back the Response message to.

The Response Table has a similar structure to the Routing Table but it serves a different purpose. In Table 3.2 it is possible to see the structure of the Response Table.

Message ID	**Next hop node (MAC address)**
Message ID #1	Device X's MAC address
Message ID #2	Device Y's MAC address
Message ID #3	Device Z's MAC address
...	...
Message ID #9	Device X's MAC address

Table 3.2: Response table example and format

Despite having similar structures, the two tables store different information: the Routing Table maps a MAC address to an hop distance estimate, whereas the Response Table maps a message identifier to a MAC address. Hence, the need of having two distinct tables.

Two main actions are allowed in a Response Table:

- To add new rows and to retrieve the next hop's MAC address for a certain message identifier. The message identifiers are random numbers ranging from -2^{31} to 2^{31}, providing around 4.3 billion possible numbers. Since the message identifiers are unique, the existing Response Table rows are never modified, only new ones are added.

- Given a message identifier, the next hop's MAC address should be returned. Since to a message identifier corresponds a single MAC address there should be no problems in the response path. However, in case the requested message identifier is not found in the Response Table, the application is notified that the response can not be routed due to the lack of a next hop MAC address.

Figure 3.13 presents an example with three devices: A, B and C. Device C has an active Internet connection. The figure also shows the routing tables of each device after the discovery and advertisement processes are completed. Device A will choose B to send a Request message and B will choose C, since they provide the best estimates, respectively.

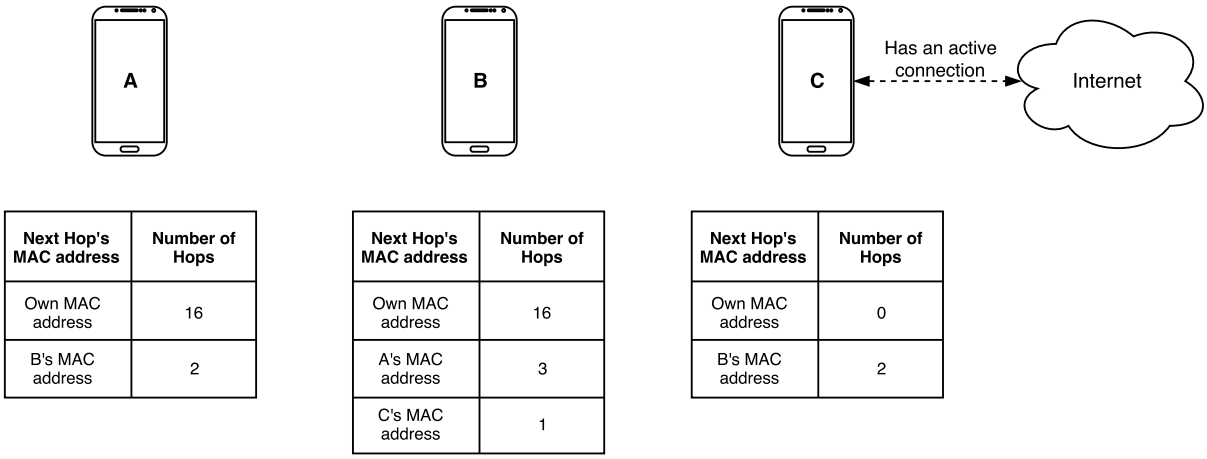


Figure 3.13: Example 2: State of the routing tables of the three devices.

3.2.3.1 Sending a Request Message

The exchange of web pages requires a number of features that need to be mentioned before further explanation:

- A text area enabling the input of a URL from the application user.
- A button for the user to notify the application that he/she has finished inputting the desired URL. This button will act as a trigger for the application to begin the request sending.
- A mechanism to manage the actions related to web pages must be created. In this thesis the method found that best fit the requirements was the creation of a *WebView* instance.
- A handler providing the link between the *BluetoothService* and the main activity, previously mentioned in the context of the advertisement process, that can communicate the bytes received by the service to the rest of the application.
- Finally, a mechanism to identify how the received bytes should be processed, *i.e.*, what format should they be converted into. This mechanism must also provide a method to parse the protocol messages and instruct the application to act accordingly, *i.e.*, to follow a specific set of actions depending on the message type.

In Figure 3.14 a flow diagram representing the request sending process is shown. It provides a visual overview of the application workflow when user input triggers the request sending mechanism.

Assuming the device already established the best route to reach the Internet, *i.e.*, its routing table is populated, the journey of the web page request from the user input to the display of the response will be explained.

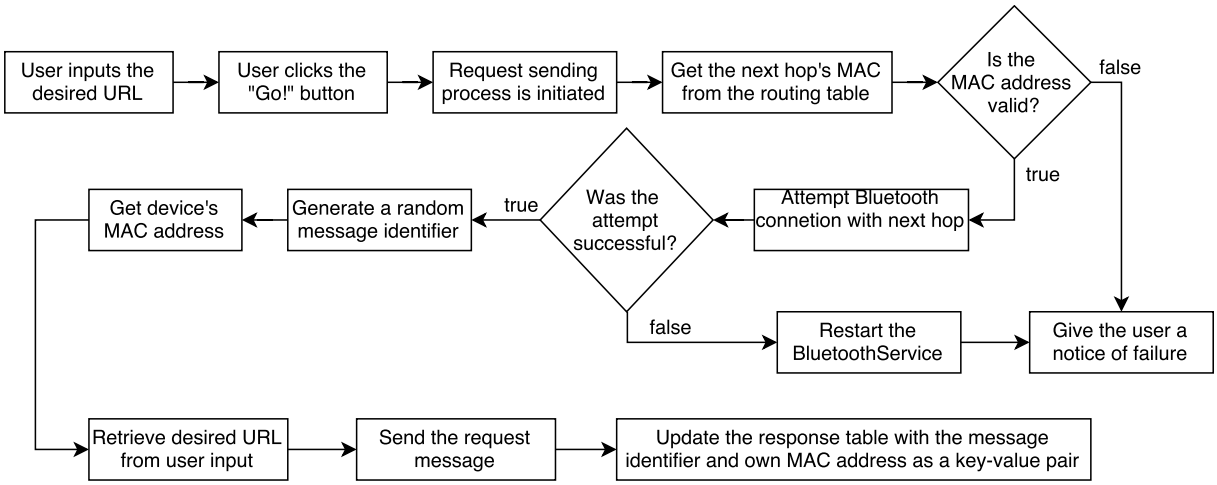


Figure 3.14: Flow diagram of the request sending process triggered by user input.

In Figure 3.15, a generic Request message is shown. Its format does not differ much from the one presented in Figure 3.7. However, two new fields are introduced: the *Message ID*, a unique identifier representing each message that will be useful to keep track of what is each message's payload and sender, and the *URL to request*, which is the URL the user is trying to access.

Type	Message ID	Bluetooth MAC Identifier	URL to request

Figure 3.15: Request message format.

The web page request process may be initiated by two different events: when the user inputs an URL and presses the ”Go!” button, or when the device receives a Request message and its next action is to forward it to the next hop.

Assuming the originating event is correctly identified and the process is aware that it was initiated by user interaction, the first action is to retrieve the next hop's MAC address, to whom the device shall send its request. This can be achieved by using the routing table retrieving the best next hop candidate to reach the Internet.

If the routing table query does not return a valid MAC address, the user is notified of the request failure. If a valid MAC address for the next hop is returned, the device attempts to establish a connection with the next hop, following the same mechanism as in the advertisement process. Should the connection attempt fail, the user is notified of the request failure, allowing him/her to restart the request process. In that case, the *BluetoothService* is also restarted to keep the device listening to incoming connections.

If the connection is successfully established, the application will proceed to the generation of the Request message to be sent.

Once the Request message is sent, the application updates its Response Table with the generated message identifier and the MAC address of the sending device, *i.e.*, its own MAC address, as a key-value pair. With this mechanism the device is able to assess if it is the destination of a certain response or if it is a relay node that should forward the response to the next hop. This assessment will be presented and explained in 3.2.3.2.

If the request sending device is forwarding a received Request message, there is no random identifier generation, since the value will be retrieved from the received Request message. Also, the Response Table is not updated because this will be done during the reception process, explained in the next subsection.

However, if the connection establishment fails, the user is not notified, since this Request message was not originated by him/her. In this case, the device queries the Response Table for the MAC address mapped by the Request message identifier, corresponding to the device that sent/forwarded the Request message to this node. A failure notice is then sent to that device, with the intent of notifying the Request message originator about the inability to forward the request until it reaches the Internet.

If the connection is successfully established, the message is sent as before, with the slight difference that the message identifier and the URL to be requested are retrieved from the values encapsulated in the received Request message – see Figure 3.15.

3.2.3.2 Receiving a Request Message

The request reception process similar to the advertise receiving. The device receives a connection attempt from a peer. If the connection is successfully established, the Request message is received by the device in *BluetoothService*.

In Figure 3.16 it is possible to see a flow diagram of the request receiving process.

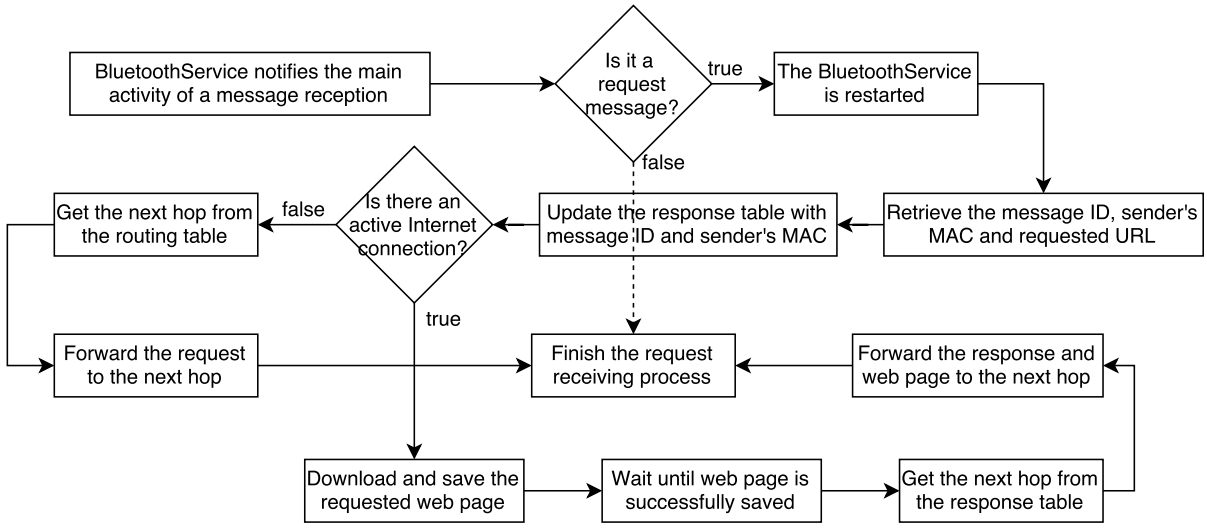


Figure 3.16: Flow diagram of the request receiving process.

As in the advertisement reception process, see Figure 3.11, the received message is classified according the possible types: an Advertisement, a Request, a Response or a Failure. The *BluetoothService* is also restarted.

Once the message is correctly classified, as an Advertisement, the message identifier and sender's MAC address are retrieved from the message – see Figure 3.7. In the request sending process it was mentioned that in case the device was not the owner of the Request message, it would not update the Response Table. For devices that receive a Request message but are not its originators, the table updating process is done during this phase. Thus, the routing table is updated with the retrieved message fields.

At this point, there are two options of execution: one regarding the devices with Internet connection and another for the devices without one. To define which approach is taken by the device, a quick check of the Internet connection status is performed. If the check returns false, meaning the device has no active Internet connection, the request will be forwarded to the next hop, retrieved from the routing table, towards a device with an Internet connection. To do this, the application proceeds as described in 3.2.3.1 for a device that is not the owner of the request, where the sent URL and message identifier are the ones that were previously retrieved from the received message and the MAC address is the device's

own address.

On the other hand, if the Internet check returns positive, it means the device is a final destination for the received request. The device will act as a communication link between the Internet and the owner of the request. It now has to retrieve the requested web page and send it backwards until it reaches its original requester.

The methodology to save the web page had several possibilities, such as saving the web page as an image, saving only the *HTML* content or saving each element of the page individually. The first two methods are not complete, meaning the web page could lose some of its features, *e.g.* dynamic images, search fields, *etc.*. The third method would fully download the web page, however it would require additional logic to save the different elements in the same directory and to arrange them to re-create the web page with its initial format. The implemented solution saves the web page as a web archive, which is specific to the *WebView* class – see [23]. It solves the problem of saving the web page's different elements and re-arranging them to restore the web page, since it downloads each element but compiles them in a web archive, that can be easily decompressed by *WebView*. Thus, it proved to be the best solution.

The thread *WaitForWebPage* (see Figure 3.1), is used to ensure that the web page is successfully saved in the application's file directory. It is specially useful for the download and saving of large web pages where this process can be time consuming. If this thread is not run, the device risks sending a Response message before having the complete web page file, failing to transmit the requested web page to its destination. This process is done concurrently, allowing the application to continue its normal functions, such as listening to incoming connections.

Once the download and saving process is completed, the device must notify the next hop about the file transfer that will occur. For this effect, a Response message is sent to the MAC address mapped by the received message identifier from the Request message. If a valid MAC address is returned, the device attempts to establish a connection and send a Response message, as explained in the next subsection.

Resuming the example in Figure 3.13 and supposing the user from device A wants to request a URL and inputs it correctly, the device will follow the steps explained in Subsection 3.2.3.1 and check the routing table to establish the next hop. Also its Response Table will be filled during this action, since it is the owner of the request.

Once that process is completed and the message reaches device B, it will update its routing table with the newly received Request message. After that, it checks its own routing table and assesses if it should forward the request or download the page. Since the device does not have an active connection, the request will be forwarded to C.

Finally, upon receiving the Request message, device C will update its own Response Table with the received message identifier and B's MAC address. Since device C has an active Internet connection, it will download the web page.

Figure 3.17 depicts this process, as well as the three Response Tables from the devices after the Request messages are all sent and the web page downloaded.

3.2.3.3 Sending a Response Message and Web Page

In this subsection the process of sending a Response message and a web page will be explained. The response will be routed back until it reaches the owner of the request that originated this response. In the previous section the process of receiving a Request message and downloading and saving the web page are explained.

In Figure 3.18 a flow diagram of the Response message and web page sending process is presented.

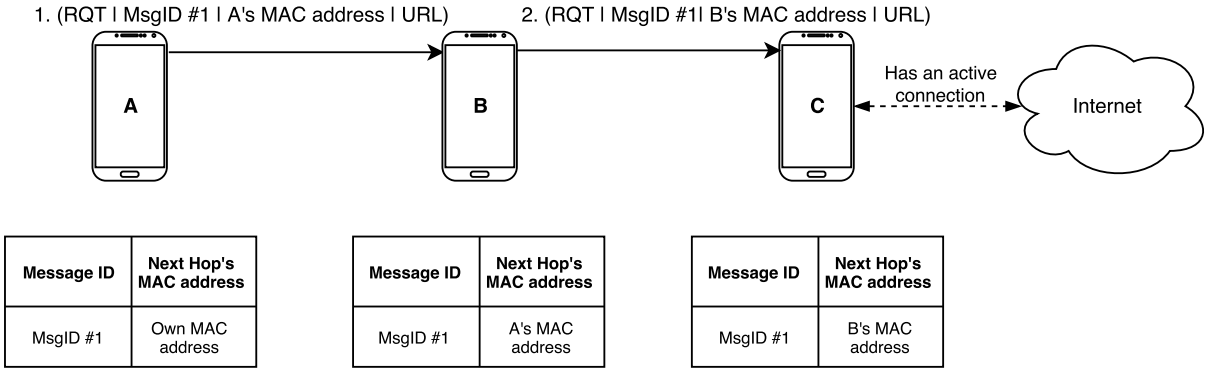


Figure 3.17: Example 2: Request sending and receiving process.

It shows the mechanism from the time the web page is saved, until it is sent. As previously described, the process is finished if no valid next hop for that message identifier is retrieved, or by correctly sending the web page as expected.

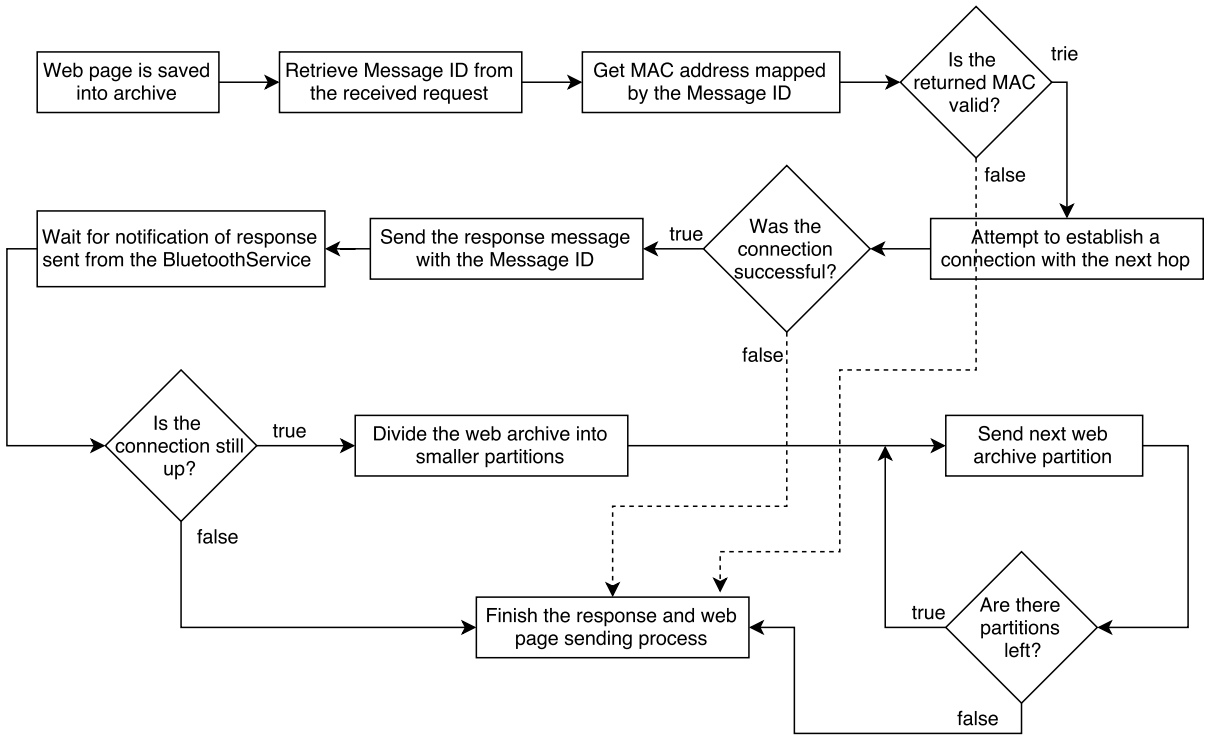


Figure 3.18: Flow diagram of the Response message and web page sending process.

To send the Response message to the correct destination, the device queries the Response Table for the MAC address of the next hop mapped by the message identifier received in the Request message. If a valid MAC address is returned, a connection is attempted, as seen in the other sending processes – see Subsections 3.2.2.2 and 3.2.3.1.

In Figure 3.19 the format of a generic Response message is shown. The *Message ID* will have a direct correspondence with the message identifier of the request that originated this response. The Response message is used as a notification, alerting the receiving device of an incoming web page file.

Once the *Main* thread is notified by the *BluetoothService* that the Response message was correctly sent and received, it must send the requested web page archive to the next hop. This notification is passed from the service to the *Main* thread via the previously mentioned handler.

Figure 3.19: Format of a Response message.

In this stage, the sending device does not need to attempt a new connection with the receiver – the established connection is maintained open until the web page transfer is complete. To initiate the file transfer, the web archive file must be converted into a byte array, due to the Bluetooth transfer mechanism explained in 3.2.1.3. The application then retrieves the archive's file size to match the array size.

The device checks if the established connection is still open. If this is the case, the bytes are then written to the input stream as usual. However, in contrast to what happened with the single text messages, the file bytes can be larger than the connection stream buffer. To overcome this problem, it is necessary to implement the segmentation of large files, *i.e.*, the ones larger than the stream buffer, into smaller partitions. These partitions must be sent individually and, during the reception, they must be regrouped into the original file, as will be shown in the next subsection.

If the connection was shut down due to unexpected behaviour from one of the parts, such as a forced disconnection due to moving out of range, the web archive is not sent, the web page sending process is aborted and the device keeps listening to incoming connections.

3.2.3.4 Receiving a Response Message and Web Page

In the receiver side, the web page and Response message are received and the device must assess if it is the final destination or if it is a relay node for a different device, in which case the web page will not be displayed, but the Response message and web page are forwarded.

In Figure 3.20 a simplified flow diagram of the Response message and web page reception process is shown. It is possible to visualize the file receiving workflow as well as the different possibilities for the receiver of the response: forwarding the response or displaying the web page. It is also shown the mechanism behind the multi-page request. With the shown implementation, the user is capable of seeing the requested web pages and navigate through those pages without having to manually send each request.

This process begins when the handler notifies the *Main* thread that a message has been received. The message is compared with the possible types and is identified as a Response message.

At this point, the device knows it will be receiving a web page archive, so the connection is not shut down as it happens in the advertise and request receiving processes – see Subsections 3.2.2.3 and 3.2.3.2, respectively. This mechanism allows the application to reduce the duration of the web page exchange.

The *BluetoothService* is notified that the next reception will correspond to a web page archive. Since the connection is still up, new incoming connections will be blocked not allowing other devices to interfere with this exchange until it is completed.

Once the message is identified and passed to the *Main* thread, it will be analysed, *i.e.*, the message identifier will be extracted – see Figure 3.7. This is done to retrieve the next hop from the Response Table. Two situations are possible: either the device is the destination of that response or it isn't and the message must be forwarded to the next hop.

The sender will then proceed with the transfer of the web page archive. At this point, the receiver of the web page is aware that a web archive is being transferred, as opposed to a text message. Using

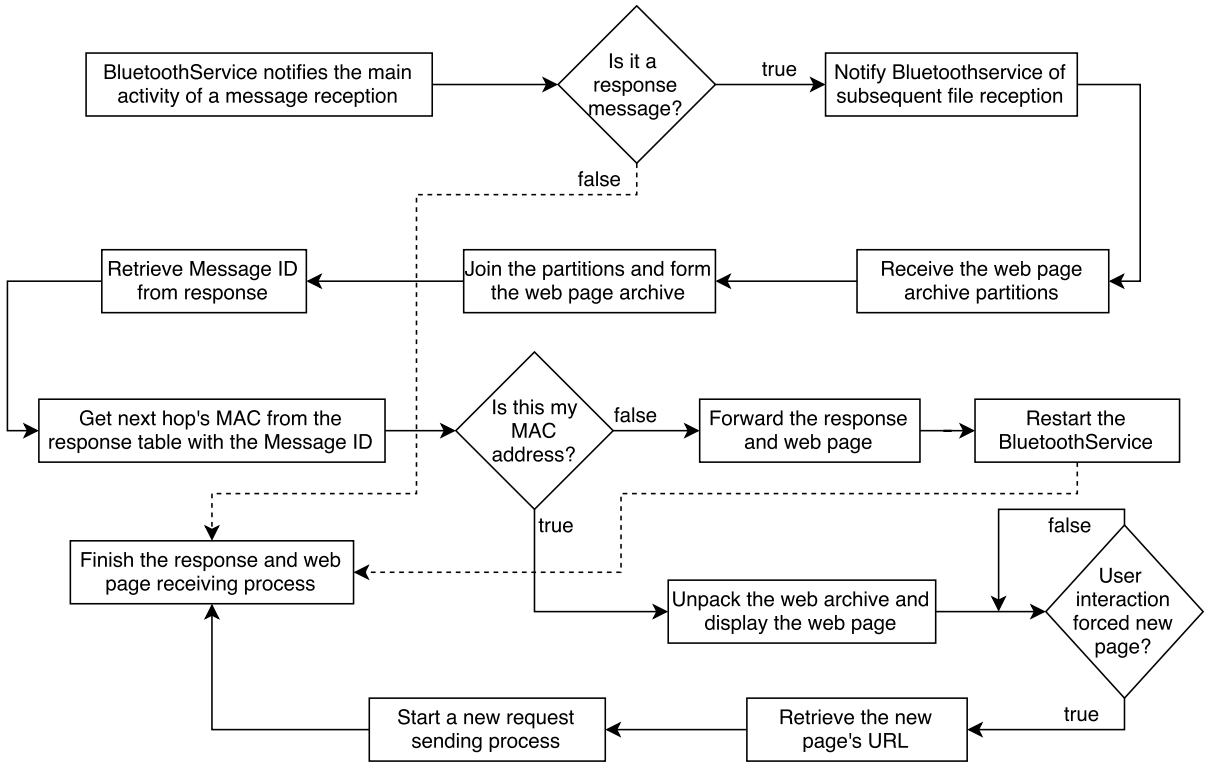


Figure 3.20: Flow diagram of the Response message and web page receiving process.

the implemented logic in *BluetoothService* for the reception of this data type, the web archive is fully received and transferred to the *Main* thread, through the handler.

Once the *Main* thread is notified about the received web archive, the device saves it in the application's directory. The service is restarted and is now able to receive new connections, changing the receiving logic to the reception of text messages, since it is not expecting to receive more segments for the time being.

The device now has received both the Response message and the web page and it has all the necessary parts to decide which action to take. That is achieved by querying the Response Table with the retrieved message identifier, from the Response message. This query returns a MAC address and the application verifies if it is valid or not.

If the MAC address is deemed valid a new check is performed, this time with the purpose of assessing if the device is the final destination of the response. If the retrieved MAC address does not correspond to its own MAC address, meaning this device is not the destination, it attempts to establish a connection with the next hop, identified by the retrieved MAC address. If the connection is accepted and established, the Response message and web page sending process is initiated – see 3.2.3.3.

On the other hand, if the device is deemed the owner of the request that originated this response, *i.e.*, its MAC address corresponds to the one retrieved from the Response Table, it means the web page archive must be unpacked and displayed to the user.

As mentioned before, the *WebView* class is used to manage the actions performed with web page and these ones are no exception. So, the *WebView* is initialized and made visible to the user. The web archive is then unpacked and displayed to the user using methods inherent to this class – see [23] to learn which ones may be useful for this purpose.

When the user receives the requested web page, he/she might want to navigate through other pages, corresponding to web links in the previous one (for instance when a user clicks in a link present in the web page, requesting a subsequent page). With the current *WebView* implementation, once a web

link is clicked, the ”no Internet connection” error will be displayed, as it would happen in a normal web browser opened in a device without Internet connection. This lack of continuity severely affects the usability of the application thus, a solution to this problem was developed.

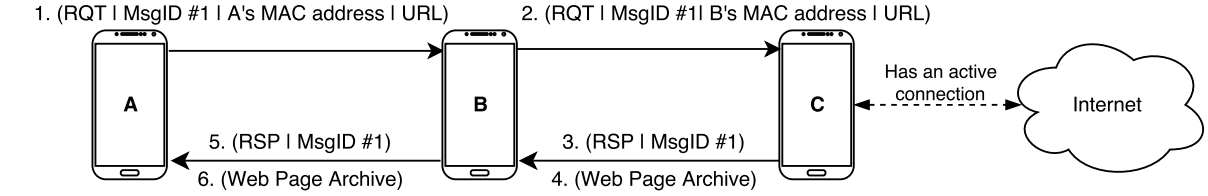
A solution that proved to be effective is to create a mechanism to detect web page loading errors, such as the ”no Internet connection” error. Once the mechanism is triggered, meaning an error occurred, the application must retrieve the requested URL and form a new Request message from it – see 3.2.3.1 for the explanation of this process. This mechanism must overwrite the usual action of simply loading the web page, when requested by the user.

With this mechanism, when the requested web page is loaded and displayed to user, he/she is free to interact with the web page as in the case of a normal web browser. Once this interaction leads to a new web page being requested and displayed, the solution’s mechanism is triggered and a new Request message is formed and sent, being submitted to the web page exchange process.

To finalize the example from Figures 3.13 and 3.17, device C, after finishing downloading the web page, will check its Response Table and send a Response message to device B, which is the next hop retrieved for that specific message identifier *MsgID #1*. The Response message is followed by the web page archive, previously downloaded, as described in Subsection 3.2.3.3.

Device B receives the Response message and the web page, following the steps previously explained in Subsection 3.2.3.4, and checks its Response Table for that message identifier. Device A’s MAC is returned from that query and that’s the destination for B’s response, so device B sends the Response message and web page to A.

Figure 3.21 illustrates this example and messages exchanged between the three devices, as well as the Response Tables, previously established in Figure 3.17.



Message ID	Next Hop's MAC address
MsgID #1	Own MAC address

Message ID	Next Hop's MAC address
MsgID #1	A's MAC address

Message ID	Next Hop's MAC address
MsgID #1	B's MAC address

Figure 3.21: Example 2: Sending and receiving of Response messages and web pages.

Chapter 4

Tests and Results

This chapter describes the experiments that were performed in order to assess the performance the developed application. The main goals are the following: to obtain an empirical and realistic conclusion about which technology serves better the purpose of this thesis and to assess how the application fares in realistic scenarios.

This chapter will be divided in two different sections: one regarding the comparison between Bluetooth and Wi-Fi Direct in Android devices, and another regarding the tests performed with the developed application, to better understand where it performs better and worse, proving or not if the theoretical choices translate into actual performance gains.

4.1 Bluetooth *vs.* Wi-Fi Direct in Android Devices

This section will cover the differences, advantages and disadvantages of both Bluetooth and Wi-Fi Direct. A series of experiments will be conducted and their results analysed, providing a justification on which technology is best suited for this application and applications with a similar architecture and/or purpose. The tests will cover the ranges of communication and discovery times of both technologies.

Since the first implementation of Bluetooth in Android, several releases have been developed. Different Bluetooth versions provide different Quality of Service (QoS), this may impact the performed tests, as a device running a newer Bluetooth version may provide better results than a device running an older version. For sake of uniformity, all the tested devices will be running Bluetooth version 4.0.

4.1.1 Communication Range

The communication ranges are of extreme importance. They can greatly reduce or increase the number of hops a packet has to pass through, in order to reach the destination. If the range of communication is too short, the number of established connections will increase, which may cause an overload of the network, and the deterioration of communication speed. On the other hand, if the communication range is long, the devices are able to reduce the number of hops, creating less traffic in the network and establishing the least possible number of connections. However, by using a longer communication range the energy consumptions will increase.

Both technologies share some similarities: they depend on the environment of the communication, *i.e.*, the elements that are surrounding the devices and possible obstacles in the way. Four different experiments are conducted, two with line-of-sight and another two without line-of-sight, having walls separating the devices, one for each communication technology. These experiments aim to provide an estimate on the real communication ranges of Bluetooth and Wi-Fi Direct, with and without line-of-sight.

The two non-line-of-sight experiments were conducted in a corridor of Torre Norte, in the vicinities of Instituto Superior Técnico. The Bluetooth line-of-sight experiment was conducted in the same location as the previous experiments. However, the Wi-Fi Direct line-of-sight experiment required a larger environment, so it was conducted in the outside area of Instituto Superior Técnico. It is important to note that, although this experiments were made with as little interference as possible, there are certain elements that are impossible to control, such as wireless communications from other devices and metal objects, such as metal lockers and cars.

Bluetooth establishes four different classes for the devices that may use this technology, depending on the transmitting power. Mobile phones are inserted in class 2 and, for that class, the specified average range of transmission in order to have a reliable connection is *10m*, from [24].

Wi-Fi Direct on the other hand offers, theoretically, ranges of communication up to, approximately, *200m* (see [25]), which represents a much better solution, in terms of network off-loading and general depth.

In order to verify these claims from both technologies, two mobile devices were taken to an open space, although with some limitations as described above, and several discovery procedures were made, until the devices stopped being discovered by one another. Both tests were made with a direct line-of-sight between devices.

Bluetooth was able to create a connection between devices from a distance up to *42m* apart (see Figure 4.1 for the overall scheme of this experiment). This value is higher than what was expected judging by the theoretical value of *10m*, although the quality of the connection was not verified – see Section 2.4 to see how the distance affects the data rates. Using Wi-Fi Direct, the devices were able to communicate from a maximum distance of *77m* (see Figure 4.2 for the overall scheme of this experiment, which is considerably lower than the theoretical value of *200m*).

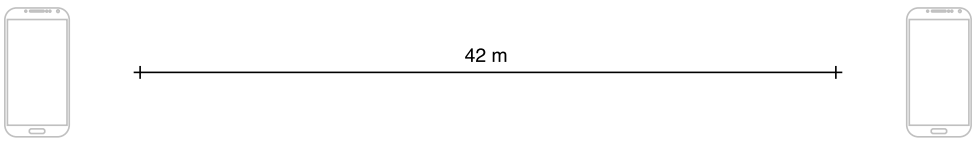


Figure 4.1: Max range of Bluetooth communication with line-of-sight between devices.

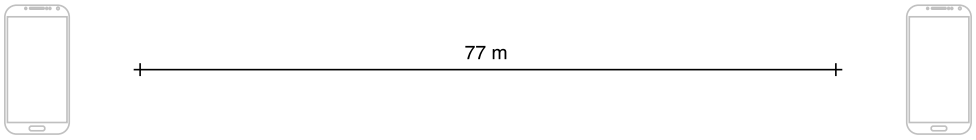


Figure 4.2: Max range of Wi-Fi Direct communication with line-of-sight between devices.

The other experiment concerns non-line-of-sight communication. It is expected that the walls standing between devices will affect significantly the communication ranges, since walls block possible signal reflections. The first test was made using Bluetooth technology where a wall was blocking the line-of-sight between devices – see Figure 4.3. The second test was made using Wi-Fi Direct, and, in order to maintain the same scenario as the previous experiment, it was located in the same place as the first – see Figure 4.4. However, due to the scenario configuration, it was impossible to recreate the experiment with only one wall, so two walls are now dividing the devices. Since the walls introduce a loss in the signal power, the more walls stand between devices, the bigger the losses will be.

As expected, the wall created a significant decrease on the maximum range of communication. The devices were able to communicate via Bluetooth from a distance of *27m*, closer to the theoretical *10m*.

Wi-Fi Direct was also able to communicate from a shorter maximum distance, measuring *29m* with

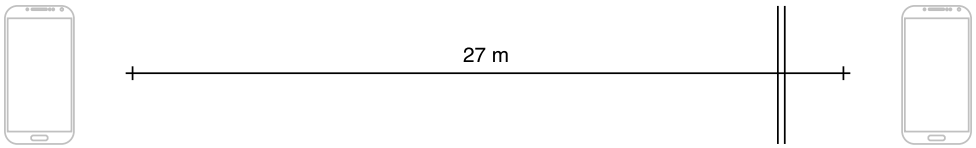


Figure 4.3: Max range of Bluetooth communication without line-of-sight between devices.

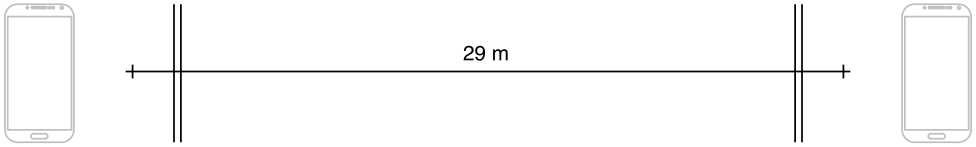


Figure 4.4: Max range of Wi-Fi Direct communication without line-of-sight between devices.

the signal passing through both walls. From a shorter distance, it was verified that this technology could communicate with only one wall in the way, meaning it also surpasses Bluetooth when an obstacle blocks the line-of-sight.

After these experiments, it is possible to conclude that Wi-Fi Direct is more advantageous, since it provides better coverage than Bluetooth to similar areas. Also, there is no evidence that Wi-Fi Direct suffers more losses from obstacles. This was already to expect, both from the theoretical values and from the transmission powers[1], since Bluetooth is characterized by lower transmit powers, when compared with Wi-Fi.

4.1.2 Discovery Times

The discovery time is a critical factor for this work. The discovery process is one of the biggest time consumers during an application run. Thus, minimizing it, is a great advantage for the overall performance of the application.

For the purpose of testing the Bluetooth and Wi-Fi Direct discovery times, three mobile devices were used: Samsung Grand Neo, running Android version 4.2.2; Motorola Moto G2, running Android version 7.1; Huawei P8 Lite, running Android version 6.0, providing an heterogeneous sample space.

Every device is running Bluetooth version 4.0, as previously stated. This Bluetooth version theoretically provides a discovery time of *10.24s*, as mentioned in [17], [27] and [28]. To confirm this hypothesis, discovery times values from the three devices were measured. Each device was submitted to multiple discoveries with a different number of discovered peers, ranging from 0 to 3.

Figure 4.5 shows the measured Bluetooth discovery times from the three tested devices. It is important to note that these results were measured with a chronometer, having a maximum precision of *0.5s*.

Samsung Grand Neo (in blue) is constant throughout the measuring, having a Bluetooth discovery time of *13s*. Motorola Moto G2 (in green) shows some variance in the various measurements. The Bluetooth discovery time ranges between *13s* and *14s*, having an average of *13.85s*. Huawei P8 Lite (in purple) shows the biggest deviation throughout the sample space. Its Bluetooth discovery time ranges between *16.5s* and *15s*, having an average of *15.976s*.

None of the three devices approached the theoretical Bluetooth discovery time value of *10.24s*. The difference is quite significant in a delay-sensitive application, such as the developed one.

Each device showed a different average Bluetooth discovery time. This difference in the discovery

[1] Transmission powers impact directly the range of transmission, since they affect the signal strength, a crucial characteristic for receivers to better capture the transmissions. A higher transmit power, usually, creates a stronger signal strength leading to the signal being capture over longer distances, as referred in [26], for instance.

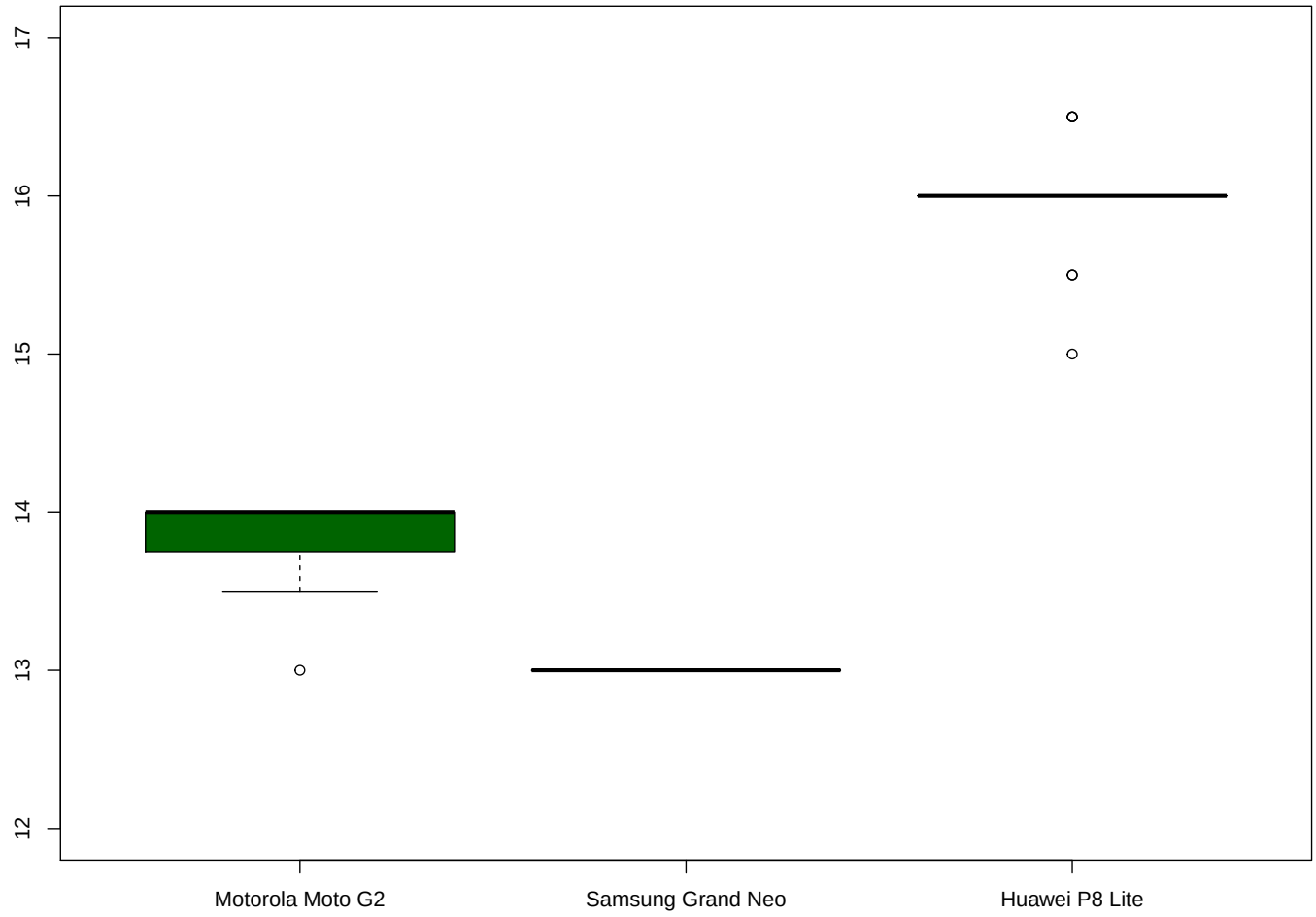


Figure 4.5: Boxplot of the measured Bluetooth discovery times from the three devices.

times can be due to the different hardware used in the devices, since each device is produced by a different manufacturer.

Wi-Fi Direct, on the other hand, does not provide a discovery time limit. This technology is constantly scanning for peer devices, until the stops the ongoing search – see [29]. This is also a cause for Wi-Fi Direct's higher energy consumption, since this process is running in background when the technology is being used.

4.2 Testing the Developed Application

This section describes a sequence of tests aiming to provide an overview of the developed application's performance. The main processes of the developed applications will be tested: the discovery, advertisement and web page exchange process.

4.2.1 Discovery Times

To obtain a meaningful analysis the devices used are the same as the ones in Subsection 4.1.2. It is expected that the measured discovery time values don't differ greatly from the ones previously obtained.

In this experiment the number of discovered peer devices also ranges from 0 to 2. However, to retrieve the Bluetooth discovery times from the developed application the Android debugger was used, providing a precision of *0.5ms*.

In Figure 4.6, the times measured during the developed application's discovery procedure are shown. Analysing the boxplot it is possible to see that (1) Samsung Grand Neo (in blue) shows the best Bluetooth discovery times, being almost constant throughout the sample space – it provides an average discovery time of *12.826s*; (2) Motorola Moto G2's (in green) discovery times were similar to Samsung Grand Neo's discovery times, except for punctual samples, where the times are slightly higher – it provides an average discovery time of *12.991s*; (3) Huawei P8 Lite (in purple) provides the most irregular measures, similarly to what happened in Subsection 4.1.2 – its measured times range from *15s* to *18s*, having an average of *16.755s*.

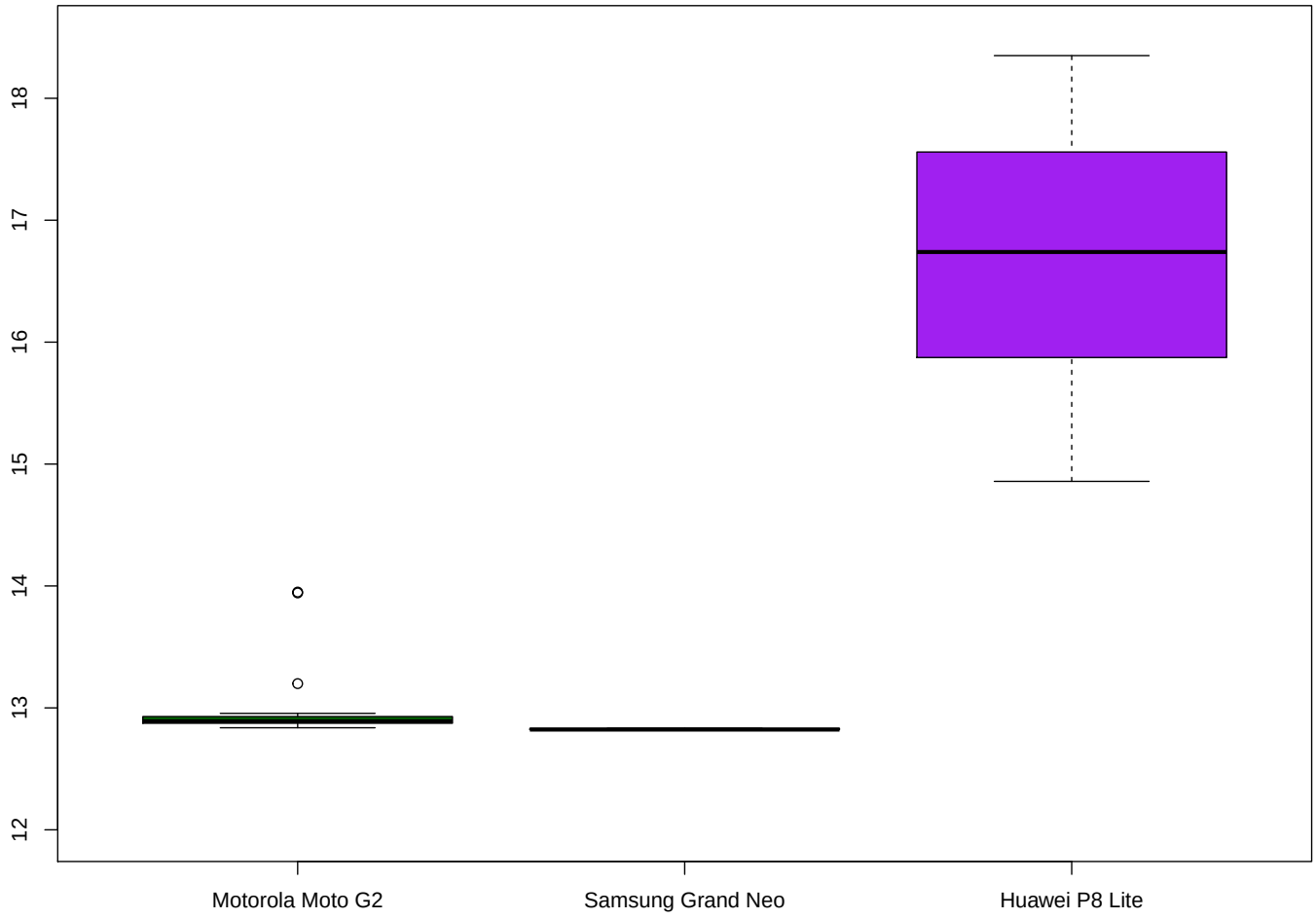


Figure 4.6: Boxplot of the Bluetooth discovery times measured while using the developed application from the three devices.

As expected, the measured application discovery times were very similar to the ones obtained during a native discovery time. All the devices maintained the same properties in the two scenarios: Samsung Grand Neo was mostly constant in both tests, providing the lowest discovery times; Motorola Moto G2 provided the second best discovery times, showing less deviation from the average in the current test than during the test performed in 4.1.2; Huawei P8 Lite showed the worst discovery times, however its average discovery time was similar in both tests. It also showed the most deviation from the average in both tests, having a larger amplitude of results in the current test, as mentioned before.

Once again, it is possible to see that all three devices failed to meet the theoretical value of *10.24s*, having shown discovery time averages of the theoretical time plus *2s* to *4s*.

4.2.2 Advertisement Times

In this subsection, the advertisement times are measured and analysed. There are two sets of tests: one where devices advertise to one peer and another one where devices advertise to two peers. These two sets of tests aim to provide a better understanding on how much time the advertisement process occupies during an application run.

The devices used to test the advertisement process duration are the same used in the previous test: Motorola Moto G2, Samsung Grand Neo and Huawei P8 Lite. Since the application needs to establish one connection per peer device found, it is expected that the advertisement times to one peer will be slightly lower than the advertisement times to two peers, although the difference should not impact the overall application runtime.

In Figure 4.7 it is possible to see a boxplot of the advertisement times of the three devices. Analysing the boxplot, it is possible to see that (1) Motorola Moto G2 has the lowest average advertisement time, as well as the lowest standard deviation from the three devices – it provides, approximately, a maximum of *2s* and a minimum of *0.75s*, having an average advertisement time of *1.333s*; (2) Unlike the tests in Subsection 4.2.1, Samsung Grand Neo has the highest standard deviation, covering the highest and lowest advertisement time values from the three devices – it provides, approximately, a maximum advertisement time of *3.5s* and a minimum of *0.35s*, having an average advertisement time of *1.514s*; (3) Huawei P8 Lite has most of its values in the interval between *1.75s* and *1s* – it provides, approximately, a maximum of *2s* and a minimum advertisement time of *0.75s*, having an average advertisement time of *1.388s*.

All three tested devices provide a similar average advertisement time. In the next test, these devices will advertise to two distinct peers. In Figure 4.8, it is possible to see a boxplot of three advertisement durations of the three tested devices. It is also visible that the disparity between durations is significantly higher than in the previous experiment, due to the supplementary complexity of advertising to more than one peer and establishing the individual connections.

In the boxplot of Figure 4.8, it can be seen that (1) Motorola Moto G2's advertisement times range from, approximately, *1.8s* to *5.2s*, having an average advertisement time of *3.066s* – in comparison with the previous test, the amplitude of advertisement durations is now of around *3.4s*, compared to the previous *1.25s*; (2) Samsung Grand Neo, similarly to the previous test, shows the biggest disparity of results, ranging from *1.7s* to *7s*, approximately, also having a higher amplitude of *6.3s*, than the previously measured *3.15s* – it provides an average advertisement time towards two peers of *3.449s*; (3) Huawei P8 Lite shows the smallest standard deviation, having measures ranging from *2.2s* to *4.1s* – this device provides an average advertisement time towards two peers of *3.414s*.

Reflecting on the results of these two experiments, it is plausible to say that, the advertisement times are independent of the device where the application is running. The advertisement times between experiments were slightly different, as was expected. Since individual connections must be established

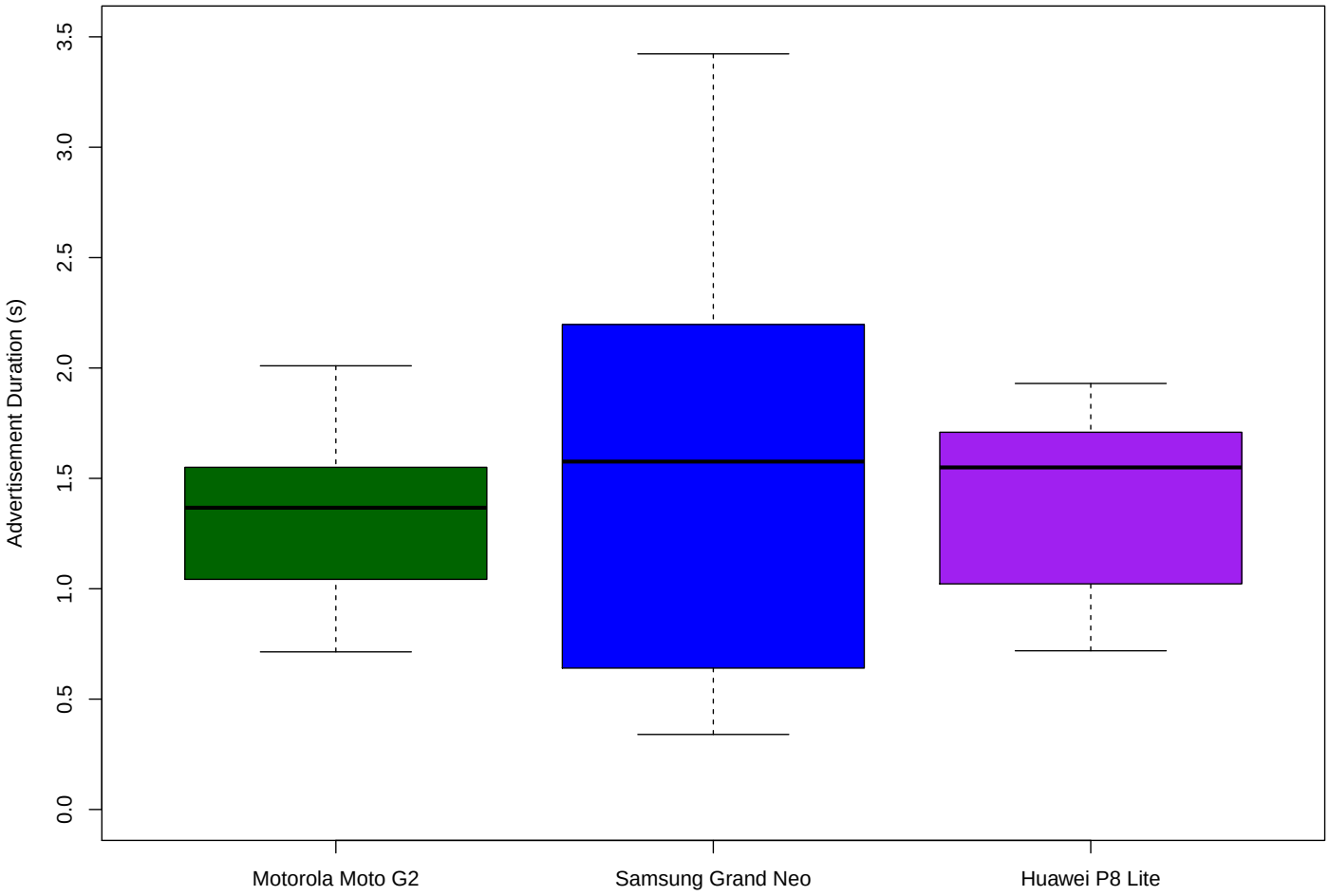


Figure 4.7: Boxplot of the advertisement times towards one peer measured while using the developed application from the three devices.

between advertiser and its peers, the first experiment sets the base advertisement time, as it reflects a one-to-one advertisement. The second experiment reflects a one-to-two advertisement and, the measured average times are double the one-to-one advertisement times, approximately.

The results of the different experiments suggest that the advertisement time is given by, approximately, the base advertisement time times the number of peers being advertised to. Although this is not confirmed, due to lack of testing equipment.

4.2.3 Web Page Exchange Throughput

In this subsection, the web page exchange data rates will be measured and analysed. Moreover, to better understand where the developed framework and application are wasting more time, an analysis of the time spent in each process will be shown.

To measure the throughput and percentage of time spent by each process, four scenarios were considered:

- The first scenario used two devices, one acting as client and another one acting as server, similar

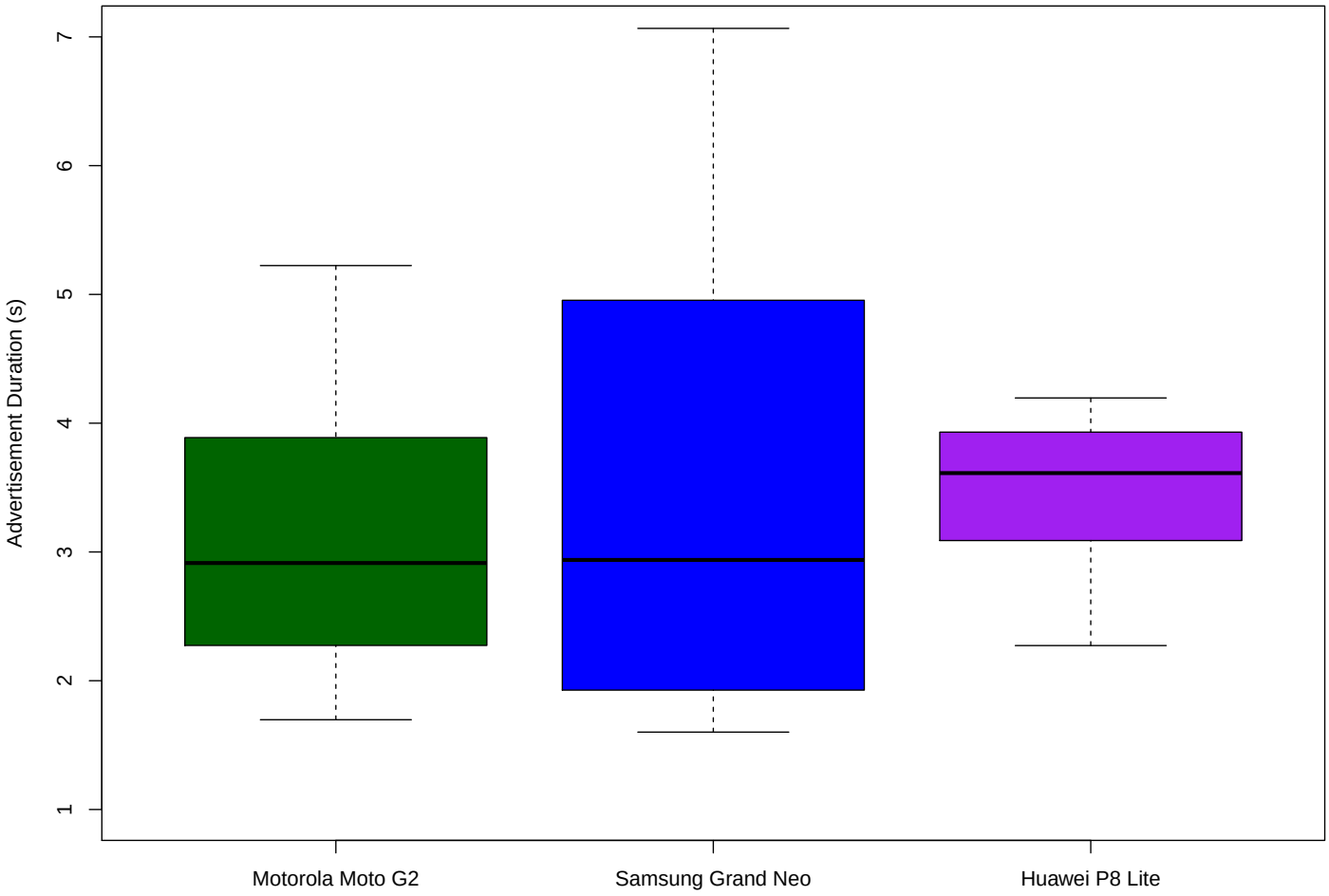


Figure 4.8: Boxplot of the advertisement times towards two peers measured while using the developed application from the three devices.

to what was shown in Figure 3.9. A web page with *1.9KB* was transferred (`https://www.example.com`).

- The second scenario maintained the same network topology. However the transferred web page was larger, sizing *105.5KB* (`https://www.google.com`).
- The third scenario used three devices, one acting as client, one as relay node and one as server, similar to the topology shown in Figure 3.13. The transferred web page was the same as in the first scenario.
- The fourth scenario used the same topology as before. However, the transferred web page corresponded to the one used in the second scenario, with *105.5KB*.

Before analysing the plots, it is important to understand what processes are present in the web page exchange, described in Subsection 3.2.3, and how was the throughput measured. Since there are two different network topologies, there will be different processes to be analysed, depending on the experiment.

In the first two experiments, two devices are used, meaning (1) one request is issued from the client to the server (*Request #1*); (2) the web page is downloaded (*Download Web Page*); (3) the response is sent from the server to the client (*Response #1*); (4) the web page is transferred from the server to the client (*Web Page Transfer #1*).

In the third and fourth experiments, three devices are used, meaning (1) a request from the client to the relay node is sent (*Request #1*); (2) the request is forwarded from the relay to the server node (*Request #2*); (3) the web page is downloaded (*Download Web Page*); (4) a response is sent from the server to the relay node (*Response #1*); (5) the web page is transferred from the server to the relay node (*Web Page Transfer #1*); (6) the reponse is forwarded from the relay node to the client (*Response #2*); (7) the web page is transferred from the relay node to its final destination (*Web Page Transfer #2*).

To measure the throughput in the first and second scenarios, the number of transferred bits are divided by the number of seconds that the application took to accomplish processes *Response #1* and *Web Page Transfer #1*.

In the third and fourth scenarios, the throughput was measured by dividing the transferred bits by the number of seconds that the application took to conclude the processes of *Response #1*, *Web Page Transfer #1*, *Response #2* and *Web Page Transfer #2*.

The *Download Web Page* and previous processes were not taken into account when measuring the throughput, as their durations are dependent on the server node's Internet connection quality.

In Figure 4.9, the measured throughputs in the previously described scenarios are shown. In both the first and third scenarios, the average throughput is approximately *0.3Mbps*. Also, if compared to the other two scenarios, it is possible to see that the standard deviations are greater. This is mainly due to the small duration of the web page exchange, making the measuring harder and less precise. The second and fourth scenarios provide similar results, with an average throughput of roughly *0.75Mbps* and lower standard deviation.

The results seen in Figure 4.9 are to be expected since, when exchanging smaller web pages, the connection and application specific computation delays, such as the queries to the Response Table to retrieve the next hop and the analysis of the received data, have a more significant impact in the overall delay than when exchanging larger web pages. Hence, the lower average throughput measured in the first and third scenarios and higher average throughput in the remaining scenarios.

If compared to the results shown in Subsection 2.4, the measured throughputs are lower than the Bluetooth indoor line-of-sight throughput measured at a distance of *0m*. In this experiment, the maximum average throughput obtained was of approximately *0.75Mbps*, a value lower than the obtained value of *2Mbps* (see Figure 2.19a). This difference is again justified with the delays introduced by the connections and application specific computations during the exchange of web pages.

To measure the percentage of time spent by each of the processes described above, the duration of that process was divided by the total duration of the web page exchange, since the initial request is sent until the final web page is received by the destination.

In Figure 4.10, a barplot is presented, showing the different percentages of time each process took, in each scenario. Starting from left to right (first to last scenario), it is possible to see that:

1. In the first scenario the majority of time was spent downloading the web page, representing over *70%* of the total duration. The remaining processes' durations sum up to less than *10%* of the total duration.

2. The second scenario shows that two processes occupy most of the total duration, *Download Web Page* and *Web Page Transfer #1*. Combined, they sum more than *90%* of the total duration, representing roughly *50%* and *40%* of the web page exchange's duration, respectively. The remaining processes are almost insignificant in the overall duration of the web page exchange.

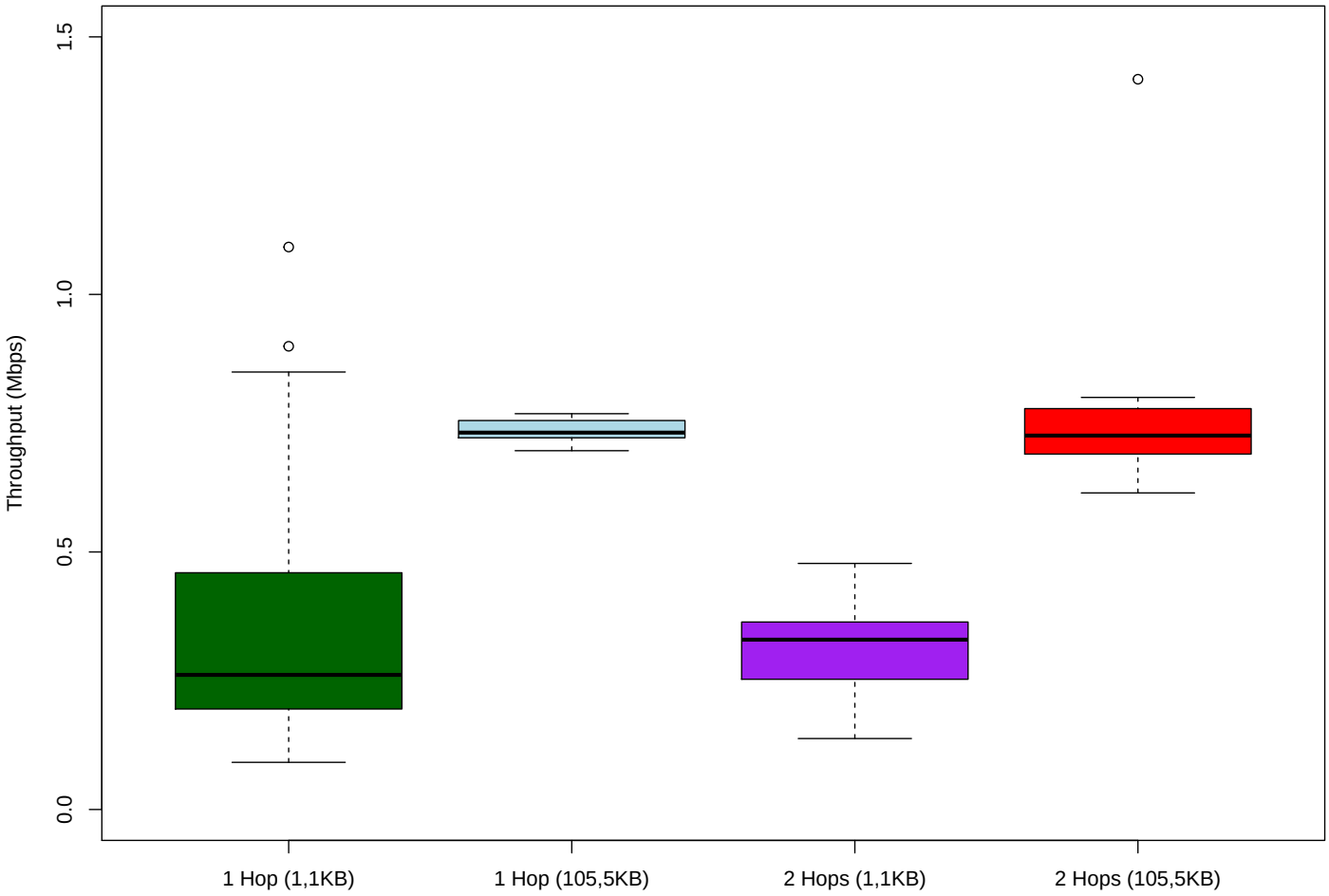


Figure 4.9: Boxplot of the measured throughput in the different experiment scenarios.

3. The third scenario shows a similar process distribution to the first scenario, with the *Download Web Page* process occupying significantly more than the rest, around *30%*. It is also seen that, despite more processes being measured, the total duration of the processes reaches less than *35%* of the total duration.

4. In the fourth scenario, the most time costly processes are *Download Web Page*, *Web Page Transfer #1* and *Web Page Transfer #2*. Together, they represent close to *70%* of the total duration. The remaining processes are insignificant, summing up to little more than *1%* of the total duration.

Figure 4.10 also shows that, in the first and third scenarios, where the web page is smaller, the majority of time is occupied downloading it, as the transfer between devices of such a small number of bytes is almost immediate. It is also possible to see that, in this scenarios, the percentage of time occupied by the measured processes is less than in the scenarios with the same topology but where a larger web page is transferred, due to the connection establishment and application specific computation delays occupying a larger portion of the web page exchange duration. In the second and fourth scenarios, these delays occupy a smaller portion of time, since the download and transfers of the web page are longer.

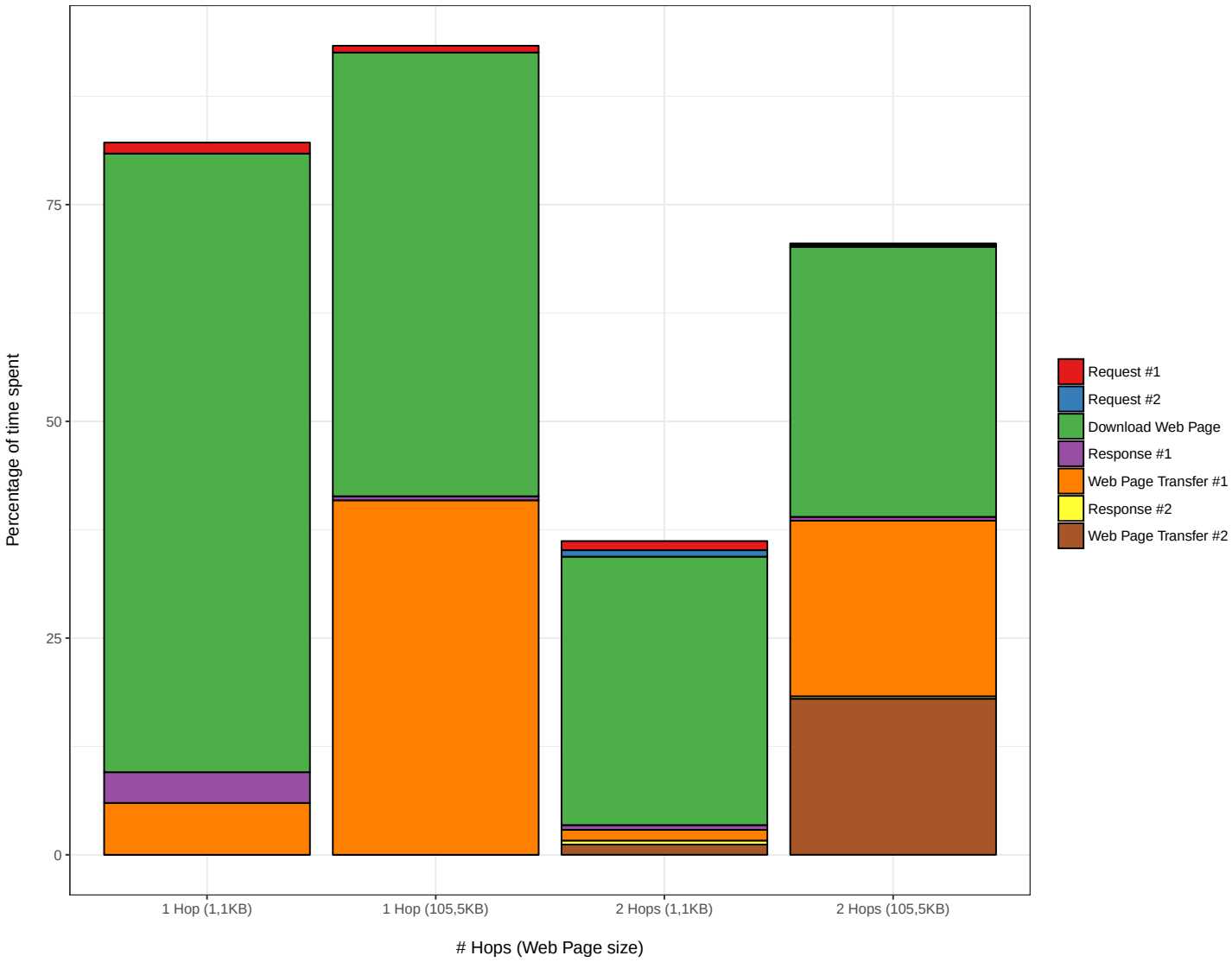


Figure 4.10: Barplot of the percentage of occupied by the various processes in the different experiment scenarios.

As expected, when the web page size is larger, so is the portion of time occupied by its transfer. The request and response sending processes are mostly insignificant, introducing little delay in the web page exchange process. Also as expected, in the third and fourth scenarios, the *Web Page Transfer #1* and *Web Page Transfer #2* processes occupy a similar percentage of the web page exchange duration, as the same number of bytes is being sent over a similar link.

As mentioned before, these results are influenced by the server node's Internet connection quality, impacting the duration the *Download Web Page* process, especially when dealing with larger web pages.

Chapter 5

Conclusions

In this chapter the conclusions of the developed work will be stated. It is divided into two sections: the first one will provide a brief summary on what was accomplished with this work. The second, regarding the limitations and possible future work of the developed framework and application.

5.1 Summary

In Chapter 2, a theoretical description of the most common cellular communication technologies was presented. Moreover, Bluetooth and Wi-Fi Direct's Android implementations were analysed, in order to provide an overview of how were these technologies used to achieve ad hoc networks and multi-hop routing within them. This analysis establishes the rough guidelines on which the developed framework would be based.

The developed framework accomplishes everything that was stated in Section 1.3. A decentralized ad hoc network can be successfully established, using mobile devices with unmodified Android system versions. Moreover, although the framework was developed using the Bluetooth technology, it is easily adaptable to other communication technologies. This is achieved due to the division of protocol/communication technology present in the framework, as can be seen in Figure 3.1. By creating the necessary links between the protocol and the communication technology, the framework is easily updated to use a different communication technology, such as Wi-Fi Direct.

The developed application also accomplishes the goals set in Section 1.3. A mechanism to download, store, transfer and display web pages was implemented. The developed application functions as intended, providing a simple but seamless web browsing experience to the user and, allowing the relay and Internet Access Point nodes to be running the application in background. Moreover, with a few tweaks, the developed application can be adapted to transfer different data formats, such as pictures, videos, or even simple messages. To accomplish this, it is needed to modify the data download and display mechanisms, adjusting them to whatever format is chosen, the remaining mechanisms can remain unchanged, as they are common to every format type.

The second objective of this thesis was to assess the advantages of Wi-Fi Direct over Bluetooth, and what the next Android system versions should be modifying to provide a meaningful Wi-Fi Direct impact, in peer-to-peer applications. This objective was fulfilled in Sections 2.4 and 4.1, where a set of study tests were performed, in order to show Wi-Fi Direct's advantages over Bluetooth.

The developed application was subject to a set of tests, aiming to assess its performance during the different processes it executes when running. The results show that the application fares the worse during the discovery process. The advertisement process suggests a linear scaling with the number

of advertised peer devices. The web page exchange process throughput was smaller than the native Android's Bluetooth throughput, but this was to be expected since other delays are introduced during this processed.

Overall, the application's performance could be increased by introducing Wi-Fi Direct as the communication technology and by transitioning the discovery and advertisement processes to a beacon advertisement and search mechanism, such as the one provided by BLE.

After the research and testing performed, it is possible to state that Wi-Fi Direct should provide an Android implementation improvement. This improvement must address the issue of user interaction when connecting two devices. If this feature is officially provided by Android, more and more applications can use Wi-Fi Direct as an alternative to Bluetooth, in Android peer-to-peer applications. Moreover, BLE's current Android implementation should also be improved, allowing Android devices to transmit and receive beacon advertisements.

5.2 Future Work

The developed framework and application accomplish the basic functions, *i.e.*, the foundation, required for a market-ready solution. However, they both have some limitations. In this section, these limitations will be analysed, explaining how future work on the developed items can tackle the current limitations.

Starting by the developed framework, it provides a easy to use set of methods with the purpose of establishing an ad hoc network. Its main limitations are the following:

- The used technology is Bluetooth, known for its low energy consumption. A better choice for the developed framework would be Wi-Fi Direct, providing (1) a larger range of communication, expanding the network reach; (2) faster data rates, reducing the time of message exchanges and overall application runtimes. Although this technology provides some advantages over Bluetooth, its current Android implementation does not allow the transmission of data without user intervention, invalidating its use in most applications.

- The implemented routing protocol is simple and light, providing an excellent solution for time sensitive applications. However, it does not support alternative path finding and data retransmission, in case of failure. In Subsection 3.1.3, a possible routing protocol is proposed, that despite being heavier, for reasons mentioned before, provides a robust solution to transmission and path failures.

- The discovery and advertisement process is slow and could be improved by using BLE's beacon technology. Devices would periodically send a beacon message, allowing peer devices to capture and store that information. This mechanism would be much more efficient than creating individual connections with each peer device to transmit the Advertisement message. However, although the used devices[1] support BLE, its current Android implementation only allows devices to capture the beacon messages, not to send them, thus invalidating the development of this mechanism.

- In this framework, security is not implemented. Despite that, it is possible to provide a rough overview of how a secure framework could be achieved (1) a central hub would be created, where Internet connected devices would send the requests; (2) a public/private key mechanism would be implemented, where every device, using an application with the developed framework, would be given the central hub's public key; (3) once a device opened the application, it would encrypt its

[1] List of devices supporting BLE's beacon advertisement: `http://altbeacon.github.io/android-beacon-library/beacon-transmitter-devices.html`.

public key with the central hub's public key, creating a sort of register mechanism; (4) all subsequent sent requests would be encrypted using the central hub's public key; (5) upon receiving a request, the central hub would decrypt the message using its private key. It would then proceed to encrypt the response data with the public key of the request originator, received in the registration process; (6) upon receiving a response to a originated request, it would be decrypted, using the device's private key.

With this security mechanism, the messages would be safely transferred between devices, without the possibility of a device decrypting data of a request it didn't originate. Furthermore, to improve the central hub's security, every stored response and request would be encrypted with a safe, well-known encryption algorithm, *e.g.* Advanced Encryption Standard (AES)-256. The central hub's storage system and overall structure is still open to debate, and could provide an interesting work to research and develop, hereafter.

The developed application provides a solid mechanism to provide web access to devices without Internet connection. Its main limitation resides in the capture and transmission of the web pages:

- The mechanism used to store the web pages is to retrieve a web page archive. This archive stores the web page in a single file, easily uncompressed and displayed. Although this appears to be the perfect solution for the web page storage, it has some flaws, inherent to the method used to retrieve the archive, *savewebarchive()*. This method's last major update was in October, 2013, with the introduction of Android API level 19 (see [30]) and, since then, web pages have suffered lots of improvements. Some of the new web page features may not be stored, due to the method's lack of compatibility thus, some web pages' interactive features may not be responsive.
- To successfully implement a full web browser experience, video streaming and file transfers should be possible. However, due to their implementation complexity, these features were left out of the initial development. These features would provide a challenging future work in this thesis, as they would include image, sound and file encryption and compression.

Bibliography

[1] W. S. Conner, J. Kruys, K. J. Kim, and J. C. Zuniga, "Overview of the amendment for wireless local area mesh networking," 2006.

[2] E. Yueh, "Android bluetooth introduction," 2009.

[3] D. Zhang, D. Zhang, H. Xiong, C.-H. Hsu, and A. V. Vasilakos, "Basa: Building mobile ad-hoc social networks on top of android," 2014.

[4] G. Hinojos, C. Tade, S. Park, D. Shires, and D. Bruno, "Bluehoc: Bluetooth ad-hoc network android distributed computing," 2011.

[5] Y. Wang, J. Tang, Q. Jin, and J. Ma, "Bwmesh: a multi-hop connectivity framework on android for proximity service," 2015.

[6] C. Casetti, C. F. Chiasserini, L. C. Pelle, C. D. Valle, Y. Duan, and P. Giaccone, "Content-centric routing in wi-fi direct multi-group networks," 2014.

[7] C. Funai, C. Tapparello, and W. Heinzelman, "Supporting multi-hop device-to-device networks through wifi direct multi-group networking," 2015.

[8] A. A. Shahin and M. Younis, "Efficient multi-group formation and communication protocol for wi-fi direct," 2015.

[9] K. Liu, W. Shen, B. Yin, X. Cao, L. X. Cai, and Y. Cheng, "Development of mobile ad-hoc networks over wi-fi direct with off-the-shelf android phones," 2016.

[10] A. Mtibaa, A. Emam, S. Tariq, A. Essameldin, and K. A. Harras, "On practical device-to-device wireless communication: A measurement driven study," 2017.

[11] Statista, "Average number of connected devices used per person in selected countries in 2014." `https://www.statista.com/statistics/333861/connected-devices-per-person-in-selected-countries/`, 2014.

[12] I. Poole, "Ieee 802.11n standard." `http://www.radio-electronics.com/info/wireless/wi-fi/ieee-802-11n.php`.

[13] S. Choudhuri, "Understanding the difference between wireless encryption protocols." `http://blogs.cisco.com/smallbusiness/understanding-the-difference-between-wireless-encryption-protocols`.

[14] M. Slob, "Multiple bluetooth connections on android." `http://profandroid.com/network/bluetooth/multiple-bluetooth-connections.html`, 2016.

[15] A. Inc., "Bluetooth technology basics." https://developer.apple.com/library/content/documentation/DeviceDrivers/Conceptual/Bluetooth/BT_Bluetooth_Basics/BT_Bluetooth_Basics.html, 2012.

[16] J. J. Garcia-Luna-Aceves, "Content-centric networking." https://www.ietf.org/proceedings/65/slides/DTNRG-12.pdf.

[17] Bluetooth Special Interest Group, "Bluetooth specification version 4.2," 2014.

[18] Wi-Fi Alliance, "How fast is wi-fi direct?." http://www.wi-fi.org/knowledge-center/faq/how-fast-is-wi-fi-direct.

[19] C. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," 1994.

[20] C. Perkins, E. Belding-Royer, and S. Das, "Request for comments: 3561," 2003.

[21] Oracle, "Class thread." http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html.

[22] G. Malkin, "Request for comments: 2453," 1998.

[23] Android Developers, "Webview." https://developer.android.com/reference/android/webkit/WebView.html.

[24] Bluetooth Special Interest Group, "How it works — bluetooth technology website." https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works.

[25] Wi-Fi Alliance, "How far does a wi-fi direct connection travel?." http://www.wi-fi.org/knowledge-center/faq/how-far-does-a-wi-fi-direct-connection-travel.

[26] A. Boukerche, "Algorithms and protocols for wireless, mobile ad hoc networks," 2009.

[27] R. Woodings, D. Joos, T. Clifton, and C. D. Knutson, "Rapid heterogeneous connection establishment: Accelerating bluetooth inquiry using irda," 2002.

[28] B. S. Peterson, R. O. Baldwin, and R. A. Raines, "Bluetooth discovery time with multiple inquirers," 2006.

[29] Android Developers, "Wi-fi peer-to-peer — discovering peers." https://developer.android.com/guide/topics/connectivity/wifip2p.html#discovering.

[30] Android Developers, "Kitkat." https://developer.android.com/reference/android/os/Build.VERSION_CODES.html#KITKAT.