

APPROVAL SHEET

Title of Dissertation: Infrastructure-less Group Data Sharing Using Smart Devices

Name of Candidate: Ahmed Amer Abdeldayem Shahin
Doctor of Philosophy, 2017

Dissertation and Abstract Approved: _____
Dr. Mohamed Younis
Associate Professor
Computer Science and Electrical Engineering

Date Approved: _____

ABSTRACT

Title of Document: INFRASTRUCTURE-LESS GROUP DATA SHARING USING SMART DEVICES

Ahmed Amer Abdeldayem Shahin, Ph.D., 2017

Directed By: Mohamed Younis, Associate Professor
Department of Computer Science and
Electrical Engineering

Advances in pervasive communication technology have enabled many unconventional applications that facilitate and improve the safety and quality of life in modern societies. Among emerging applications is situational awareness where individuals and first-responders receive timely alerts about serious events. Another example is exchanging road conditions between vehicles in a peer-to-peer fashion. The increasing popularity of smart devices and their support for multiple device-to-device (D2D) communication standards have made them an attractive choice for realizing these emerging applications. However, most existing protocols for data sharing among smart devices either require an internet connection, which may not be available, could incur extra costs, or suffer from the device's operating system limitations. Moreover, there is no existing solution that allows a set of devices to start sharing data dynamically without forcing users to apply an elaborate procedure for setting up a group. These shortcomings render existing solutions unsuitable for emergency cases and highly dynamic applications.

In this dissertation, we fill such a technical gap and present a framework for enabling an infrastructure-less data exchange in a cost-effective and timely manner

through the establishment of peer-to-peer links among smart devices. In addition, our framework opts to minimize the required user interaction for setting up a connection. Our framework consists of a suite of protocols for data exchange using Wi-Fi Direct. First we present a protocol for Alert Dissemination using Service discovery (ADS) in Wi-Fi Direct that is suitable for short messages. ADS uses the service discovery feature of Wi-Fi Direct for distributing its data in a connectionless manner, thus avoiding the setup delay in creating Wi-Fi Direct groups. In addition, we present an Efficient and Lightweight protocol for peer-to-peer Networking of smart devices over Wi-Fi Direct (ELN) that is suitable for sharing large amounts of data between a group of users. ELN mainly provides a group management solution that allows dynamic memberships and adapts for topology changes. Finally, we present an Efficient Multi-group formation and Communication (EMC) protocol for Wi-Fi Direct that is suitable for sharing data between many users distributed along a wide area, which cannot be covered by one group. EMC allows potential group owners to be qualified based on certain criteria and enable dynamic formation of groups. Moreover, EMC allows data exchange between different Wi-Fi Direct groups. To support the implementation of EMC in Android, we have developed an IP Subnet Negotiation Protocol for Seamless Multi-Group Communications (ISNP). ISNP overcomes a limitation of Android's Wi-Fi Direct implementation that forces all the formed groups to share the same range of IP addresses.

All the proposed protocols have been validated through implementation on actual Android devices. In addition, the performance of our framework in large setups is studied through simulation, where a library for Wi-Fi Direct and our protocols has

been developed and added to OMNet++. To the best of our knowledge, our framework is the first comprehensive peer-to-peer solution for mobile devices that takes advantage of the capabilities of Wi-Fi Direct.

INFRASTRUCTURE-LESS GROUP DATA SHARING USING SMART
DEVICES

By

Ahmed Amer Abdeldayem Shahin

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2017

© Copyright by
Ahmed Amer Abdeldayem Shahin
2017

Dedication

I dedicate this to my parents, my wife, and my kids.

Acknowledgements

First, all thanks are due purely to Allah, for providing me the blessings and the strength to complete this dissertation. Second, I would like to express my deepest appreciation to my advisor Dr. Mohamed Younis, for his academic and personal help and support during my PhD journey. He is always ready to provide advice, share his deep and vast knowledge, and persistently encourage me while working. Moreover, I would like to thank Dr. Charles Nicholas, Dr. Chintan Patel, Dr. Tinoosh Mohsenin, and Dr. Waleed Youssef for serving on my dissertation committee and for their valuable inputs. The thanks are also due to the members of the ESNET lab for their friendship, collegiality and inspiration.

I would like to acknowledge my father, Amer, and my mother, Attiat, for their continuous support, encouragement, and prayers. Furthermore, I would like to thank my wife, Aya, for her patience, her great support, and her understanding during such difficult time. If I am to share this PhD with anyone, it would be her. I acknowledge also my lovely kids Yahia, Safia, and Yamen for being beside me during my study.

I would like also to acknowledge the Cultural Affairs & Missions Sector at the Egyptian Ministry of Higher Education for financially supporting me for four years during my study. Finally, it is difficult to individually acknowledge everyone who has helped me to reach this degree. Therefore, I would like to thank everyone who has provided support and encouragement during my study at the University of Maryland, Baltimore County.

Table of Contents

Dedication	ii
Acknowledgements	iii
Table of Contents	iv
List of Tables	vii
List of Figures	viii
Chapter 1: Preliminaries and Research Problem	1
1.1 Introduction	1
1.2 Device-to-Device Technologies	2
1.2.1 Bluetooth	3
1.2.2 Wi-Fi Ad-hoc Mode	4
1.2.3 Wi-Fi Direct	4
1.3 Research Goals	11
1.4 Research Contribution	11
1.5 Organization	13
Chapter 2: Related Work	15
2.1 Data Sharing using Wi-Fi Direct	15
2.1.1 The applicability of Wi-Fi Direct	16
2.1.2 Intra-Group Data Sharing in Wi-Fi Direct	17
2.1.3 Inter-Group Data Sharing:	20
2.2 Data Sharing using other Technologies	21
2.2.1 Data Sharing in VANETs	21
2.2.2 Data Sharing in Mobile Devices/Networks	23
2.2.3 Data Sharing in Ad-hoc Networks	24
Chapter 3: Alert Dissemination Protocol Using Service Discovery in Wi-Fi Direct ..	26
3.1 Problem Statement and Solution Strategy	26
3.2 ADS Protocol	28
3.2.1 Service Record	29
3.2.2 Local Alert Management	31
3.2.3 Managing Remote Alerts	32
3.3 ADS Implementation	34
3.4 Performance Analysis	37
3.4.1 Alert Reporting	37
3.4.2 Alert Pruning	39
3.5 Conclusions	40
Chapter 4: Efficient P2P Networking of Smart Devices over Wi-Fi Direct	41
4.1 ELN Approach	41
4.1.1 Connection Establishment Phase	42
4.1.2 Group Management Phase	44
4.2 Implementation and Validation	50
4.2.1 Android Implementation Issues	51

4.2.2	Remote Streaming of Sensors Readings	52
4.2.3	Group Chatting.....	53
4.3	Performance Evaluation.....	58
4.3.1	Protocol Overhead	59
4.3.2	Topology changes	60
4.4	Conclusions.....	62
Chapter 5: Efficient Multi-Group Formation and Communication Protocol for Wi-Fi Direct.....		64
5.1	Approach Overview	65
5.1.1	Support of Initial Data Exchange:.....	66
5.1.2	Support of Intra and Inter Group Communication.....	70
5.1.3	Insuring Network Connectivity.....	70
5.2	EMC Protocol	73
5.2.1	Choosing Proposed GOs	74
5.2.2	Creating Groups	75
5.2.3	Selecting a Group to Join	76
5.2.4	Selecting Proxy Members	77
5.2.5	Teardown and restart.....	78
5.3	EMC Implementation.....	80
5.3.1	Wi-Fi Direct Multi-Group Chat Application for Android	80
5.3.2	Android Framework Modifications	81
5.3.3	Test Cases	82
5.4	Performance Analysis	85
5.4.1	Group Formation.....	86
5.4.2	Multi-Group Communication	87
5.5	Conclusion	87
Chapter 6: IP Subnet Negotiation in Wi-Fi Direct for Seamless Multi-Group Communications.		89
6.1	Problem Statement.....	90
6.2	The ISNP Protocol	91
6.2.1	ISNP Overview	91
6.2.2	Application-Level Module.....	93
6.2.3	OS-Level Module.....	97
6.3	Implementation and Testing	98
6.3.1	Response Time Performance	101
6.3.2	Subnet Conflict Evaluation	102
6.3.3	Integration with EMC	104
6.4	Conclusion	104
Chapter 7: Simulation Experiments		106
7.1	Building a Simulator for Wi-Fi Direct.....	106
7.1.1	Tools Used for the Simulator.....	107
7.1.2	Implementing the Simulator	107
7.2	Experiment Setup.....	111

7.3	Performance Metrics	114
7.3.1	Connectivity.....	115
7.3.2	Response Time.....	116
7.3.3	Messaging Overhead.....	116
7.3.4	Power Consumption.....	116
7.3.5	Subnet Conflicts.....	117
7.4	Simulation Results	117
7.4.1	Performance of Integrated EMC	117
7.4.2	The Effect of Parameters	134
7.5	Conclusions.....	165
Chapter 8: Conclusions and Future Work.....		166
8.1	Summary of Contribution	166
8.2	Future Work	169
8.2.1	Routing Data Between Groups	169
8.2.2	Secure Data Sharing Between Devices.....	170
8.2.3	Incorporating Other D2D technologies.....	170
8.2.4	Extend our Work to Other Platforms	170
References.....		172

List of Tables

Table 1-1 Comparison of different power classes in Bluetooth	3
Table 3-1 Application components and their description	35
Table 4-1 Application components and their description.	56
Table 4-2 The assumptions used in this section.....	59
Table 6-1 The specifications of the devices involved in testing.....	100
Table 7-1 List of the paramters used for the simulation	112
Table 7-2 The relation between the different performance metrics and out protocols.	114
Table 7-3 The mobility parmaeters used to model person movement.....	143
Table 7-4 The effect of paramters on Performance	149

List of Figures

Figure 1-1 An example of a Wi-Fi Direct group	5
Figure 1-2 Virtual interfaces required for concurrent operation in Wi-Fi Direct	7
Figure 3-1 Example usage of ADS.	28
Figure 3-2 The format of the alert record stored on the device.	29
Figure 3-3 A flow chart description of the ADS protocol.	31
Figure 3-4 Screenshots from two devices showing local alerts (marked red) and remote alert (marked blue) from the other device.....	36
Figure 3-5 A graph showing the actions taking to report an alert.	38
Figure 4-1 The connection establishment phase from the sender side.	43
Figure 4-2 The connection establishment phase from the receiver side.	44
Figure 4-3 The topology for the management connections.	45
Figure 4-4 The topology for the data exchange connections.	46
Figure 4-5 The flowchart for the group management protocol.....	50
Figure 4-6 A device is receiving and displaying the sensor readings.	53
Figure 4-7 Three devices are chatting together.	58
Figure 5-1 A typical topology for network after running EMC	65
Figure 5-2 The format of the DeviceInfo and the SAP records	66
Figure 5-3 Devices Ranks as seen by reachable devices	68
Figure 5-4 Three Wi-Fi Direct groups need to have PMs to be able to connect to each other. Devices with dark shade denote GOs. Each group has its shape symbol.....	71
Figure 5-5 An example of a cost matrix where the minimum cost assignments for each task are colored green.	72
Figure 5-6 An example of a cost matrix that a certain GO could have. On the left, we see that certain GMs can cover more than one group, thus we see in their rows the same cost repeated. On the right, we see the missing entries in the matrix been fixed by adding a very small value.	73
Figure 5-7 A state diagram of EMC	74
Figure 5-8 Psedue code for selecting candiate GO step.	75
Figure 5-9 Psedue code for creating groups step.	76
Figure 5-10 Psedue code for selecting groups step.....	77
Figure 5-11 Psedue code for selecting proxy members step.....	78
Figure 5-12 Psedue code for teardown step.	79
Figure 5-13 Screen capture of two devices running EMC.....	81
Figure 5-14 Analysis of wireless interference observed during the test.....	83
Figure 6-1 Two adjacent Wi-Fi Direct groups sharing the same IP subnet.....	89

Figure 6-2 IP subnets for two adjacent groups after integrating ISNP with EMC	91
Figure 6-3 The integration between the two part of ISNP and Android.....	92
Figure 6-4 The new format of EMC's DeviceInfo record that is used by ISNP	93
Figure 6-5 Resolving conflicts on ISNP.	95
Figure 6-6 Pseudo code for the application part of ISNP	96
Figure 6-7 Screen shots of ISNP on Nexus4 and LG Optimus Fuel	99
Figure 6-8 Average response time of ISNP with device count.....	101
Figure 6-9 Average response time of ISNP per device.....	101
Figure 6-10 Average response time of ISNP with device count.....	103
Figure 6-11 Average response time of ISNP per device.....	103
Figure 7-1 The internal design of a Wi-Fi Direct Host.....	110
Figure 7-2 An example of the static grid deployment	118
Figure 7-3 An example of the stationary connected graph deployment	119
Figure 7-4 The connectivity results from GEMC, GBAT, and GRND in case of Static Grid topology	122
Figure 7-5 The response time results from GEMC, GBAT, and GRND in case of Static Grid topology	123
Figure 7-6 The overhead results from GEMC, GBAT, and GRND in case of Static Grid topology	124
Figure 7-7 The power consumption results from GEMC, GBAT, and GRND in case of Static Grid topology.....	125
Figure 7-8 The connectivity results from GEMC, GBAT, and GRND in case of Stationary Connected Graph topology.....	128
Figure 7-9 The response time results from GEMC, GBAT, and GRND in case of Stationary Connected Graph topology.....	129
Figure 7-10 The overhead results from GEMC, GBAT, and GRND in case of Stationary Connected Graph topology	130
Figure 7-11 The power consumption results from GEMC, GBAT, and GRND in case of Stationary Connected Graph topology	131
Figure 7-12 The connectivity results from MUNK, FRST, and PRND in case of Static Grid topology	133
Figure 7-13 The connectivity results from MUNK, FRST, and PRND in case of Stationary Connected Graph topology.....	134
Figure 7-14 The effect of changing TxPower on connectivity	136
Figure 7-15 The effect of changing TxPower on response time.....	137
Figure 7-16 The effect of changing TxPower on Overhead	138
Figure 7-17 The effect of changing TxPower on power consumption	139
Figure 7-18 The effect of changing PathLoss on connectivity	141

Figure 7-19 The effect of changing PathLoss on Overhead	141
Figure 7-20 The effect of changing PathLoss on power consumption	142
Figure 7-21 The effect of Mobility on connectivity	146
Figure 7-22 The effect of Mobility on response time	147
Figure 7-23 The effect of Mobility on overhead	148
Figure 7-24 The effect of Mobility on power consumption	149
Figure 7-25 The effect of changing $T_{\text{sendInterval}}$ on connectivity	151
Figure 7-26 The effect of changing $T_{\text{sendInterval}}$ on overhead	152
Figure 7-27 The effect of changing $T_{\text{sendInterval}}$ on power consumption	153
Figure 7-28 The effect of changing $T_{\text{declareGO}}$ on connectivity.....	155
Figure 7-29 The effect of changing $T_{\text{declareGO}}$ on overhead.....	156
Figure 7-30 The effect of changing $T_{\text{declareGO}}$ on power consumption.....	157
Figure 7-31 The effect of changing T_{selectGO} on connectivity	158
Figure 7-32 The effect of changing T_{selectGO} on overhead	158
Figure 7-33 The effect of changing T_{selectGO} on power consumption	159
Figure 7-34 The effect of changing $T_{\text{HeartBeatGM}}$ on connectivity	160
Figure 7-35 The effect of changing $T_{\text{HeartBeatGM}}$ on overhead.....	161
Figure 7-36 The effect of changing $T_{\text{HeartBeatGM}}$ on power consumption.....	162
Figure 7-37 The effect of changing $T_{\text{HeartBeatGO}}$ on connectivity	163
Figure 7-38 The effect of changing $T_{\text{HeartBeatGO}}$ on overhead.....	164
Figure 7-39 The effect of changing $T_{\text{HeartBeatGO}}$ on power consumption	164

Chapter 1: Preliminaries and Research Problem

This chapter provides the motivation and necessary preliminary discussion about peer-to-peer data sharing between devices without relying on the communication infrastructure. The chapter also highlights the main research problem which is addressed in the dissertation and summarizes the contribution.

1.1 Introduction

Recent advances in pervasive communication technology have been leveraged in many unconventional applications that facilitate and improve the safety and quality of people's life in modern societies. An example of these applications is situational awareness where people exchange alert information with emergency units regarding an emerging event, such as a natural disaster [1]-[12]. Telecommunication infrastructure such as cellular towers and Wi-Fi access points may be down at that time. Another example is when exchanging road conditions between peer-to-peer networked vehicles without the involvement of roadside units [13]-[18]. The wide-spread of smart portable devices such as iPhone, iPad, Android phones, and Android tablets has made them an attractive venue for realizing these emerging applications. These devices support technologies such as Bluetooth, Wi-Fi ad-hoc mode and Wi-Fi Direct that enable them to communicate without the need for infrastructure. Thus, they can perform Device-to-Device (D2D) data exchange at an increased level of convenience. In addition, most of these devices are equipped with sensors such as accelerometer, gyroscope, barometer,

light, pedometer, etc., that can provide a wealth of information about the surroundings once their readings are aggregated.

However, most existing protocols for data sharing among smart portable devices either require an internet connection [19]-[23], which may not be available, may incur extra charges, or suffer from the device's operating system (OS) limitations. In addition, there is no existing solution that allows a set of devices to start sharing data dynamically without forcing the users to apply an elaborate procedure for setting up a group. These shortcomings render existing solutions unsuitable for emergency cases and highly dynamic environments.

1.2 Device-to-Device Technologies

Many researchers has proposed solutions for D2D over cellular networks or Wi-Fi networks [24]-[31]. However, these solutions require (assume) the availability of infrastructure, such as cellular towers and access points, to allow the devices to communicate. In certain rural areas or areas suffered from natural disasters, there may be no coverage from neither Wi-Fi nor cellular networks. In addition, routing communication traffic through a Wi-Fi network or a cellular network may introduce unnecessary delays, power consumption, or costs, especially when the devices are in the proximity of each other.

Bluetooth, Wi-Fi Ad-hoc mode, and Wi-Fi Direct are example technologies that do not require any infrastructure support. Most of the smart devices nowadays that have the necessary hardware and software to support any of these technologies can communicate directly in a way that reduces latency, cost and power consumption, and

thus make it possible to share data between groups of devices. An example of a commercial solution that benefit from these technologies is Google NearBy [32], which have a set of APIs that allow for discovering nearby devices using Wi-Fi Direct, Bluetooth, and the acoustic signals. However, such a solution still depends on the cloud for completing the discovery procedure and for doing the actual data transmission. Thus, it suffers from the problems we mentioned earlier. The balance of this section review the basic features and compares these popular technologies.

1.2.1 Bluetooth

Bluetooth is one of the most popular technologies in smart devices. The most recent version of the Bluetooth specification is V4.2 [33] which added Low Energy (LE) capabilities. Bluetooth basic data rate is 3 Mbps, but there is a version of Bluetooth called High Speed that can reach 25 Mbps by using 802.11 networks to do the actual data transfer. The range of the Bluetooth depends on the power class of the device. Basically, there are three power classes. Table 1-1 [1] summarizes the different power consumptions and ranges for the different classes. Typically, mobile phones are considered Class 2 devices.

Table 1-1 Comparison of different power classes in Bluetooth

	Range (m)	Power Consumption (mW)
Class 1	100	100
Class 2	10	2.5
Class 3	1	1

Bluetooth LE allows a device to work as a beacon to advertise data. Other Bluetooth LE devices can receive these advertisements without the need for prior connections.

The most restraining features of Bluetooth are its range and bandwidth, which limit its use for D2D communications.

1.2.2 Wi-Fi Ad-hoc Mode

Wi-Fi Ad-hoc mode is part of the IEEE 802.11 specifications that exists along with the infrastructure mode. In contrast to Wi-Fi Infrastructure mode, the Ad-hoc mode allows devices to communicate without intermediate access points by forming a dynamic peer-to-peer (P2P) network. To form the network, the devices must be configured to use the same setup, such as SSID, operating channel, etc. Although Ad-hoc Wi-Fi initially gained a lot of interest [35]-[41], its use has become limited nowadays due to some shortcomings. Among the most notable shortcomings is that Ad-hoc Wi-Fi cannot be used concurrently with normal Wi-Fi connections. Also, the speed and range of Ad-hoc Wi-Fi are less than what can be achieved using the infrastructure mode. In addition, current smart devices do not support it out of the box. Certain modifications should be made to allow devices to use the Wi-Fi Ad-hoc mode. Due to these limitations and the uneasy setup required for enabling data sharing, Wi-Fi Ad-hoc is not suitable for D2D data sharing, especially when high speed or large scale communications are desired.

1.2.3 Wi-Fi Direct

Wi-Fi Direct (sometimes called Wi-Fi P2P) [35] is geared toward the same range and bandwidth of the normal Wi-Fi networks. Depending on the technology used (e.g., 802.11n, 802.11g, etc.), the bandwidth can reach 250 Mbps. The first version for Wi-Fi Direct specification was published in 2009 by Wi-Fi Alliance and focused on the

required specifications for allowing existing 802.11 hardware to adapt the technology. Nowadays most manufactured smart devices have support for Wi-Fi Direct; for example, Android devices starting from version 4 (Ice Cream Sandwich) have native support for Wi-Fi Direct. Given the range, and the speed, it is considered one of the best solutions for carrying out D2D communication. Since in this dissertation we employ Wi-Fi Direct as the underlying technology for D2D communication, we provide an overview of its capabilities and operation.

1.2.3.1 Basic Operation

Wi-Fi Direct enables forming groups for D2D data exchange without the need for intermediate access points (APs), as shown in Figure 1-1. Typically, one of the devices

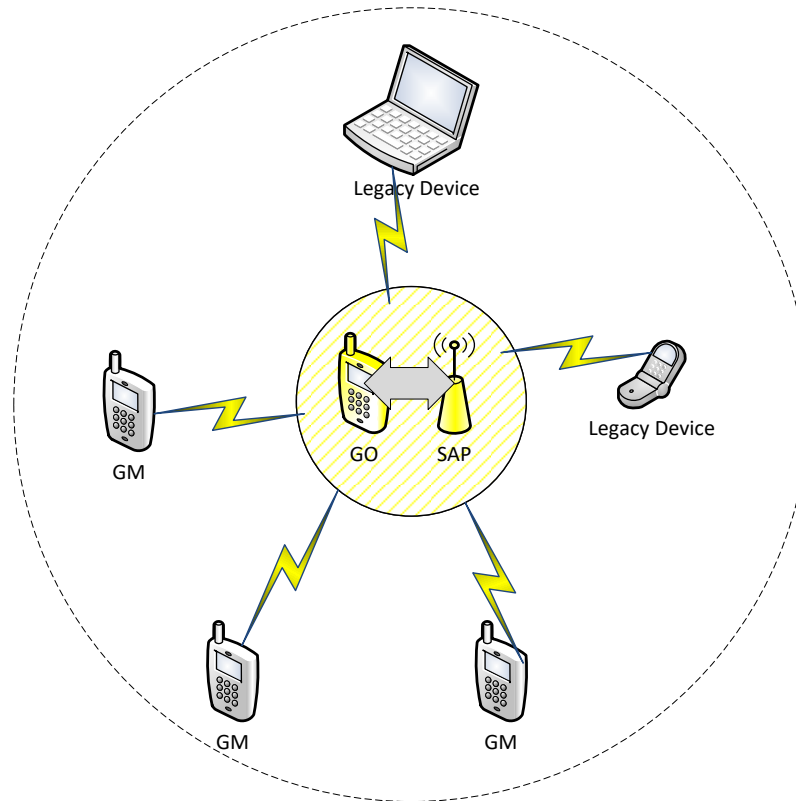


Figure 1-1 An example of a Wi-Fi Direct group

that are willing to exchange data acts as a Software Access Point (SAP) to the other devices in the group; this device is called the group owner (GO). Other devices that support Wi-Fi Direct associate with the GO using Wi-Fi Protected Setup (WPS), which resembles pairing Bluetooth devices, and become group members (GMs). Several WPS configurations such as Push Button Configuration (PBC), Label Pin, Display Pin, and Keypad Pin can be supported depending on the device capabilities. A legacy Wi-Fi device that does not support Wi-Fi Direct can become a GM in a group by associating with the SAP that is created by the GO given that it knows the WPA2 credentials of the SAP (SSID and Key).

1.2.3.2 Concurrent Operations

Basically, Wi-Fi Direct uses the Wi-Fi transceiver on the device for its operation. If concurrent connections are allowed, a device can use the same Wi-Fi transceiver to connect to a Wi-Fi Direct group and a WLAN (i.e. associated with an AP) at the same time using two different wireless channels. Thus, a device can have an internet connection while connecting to the group. Some devices are not able to do concurrent connections, so in such a case these devices must disconnect from any WLAN before being able to connect to a Wi-Fi Direct group. Concurrent connections require support from both the operating system and the Wi-Fi transceiver, for running two different MAC entities at the same time. In addition, there should be two virtual interfaces (e.g., “wlan” and “p2p”) on the device associated with the same physical interface of the Wi-Fi transceiver as shown in Figure 1-2. The “wlan” interface is used for connecting to the WLAN and the “p2p” interface is used for connecting to the Wi-Fi Direct group.

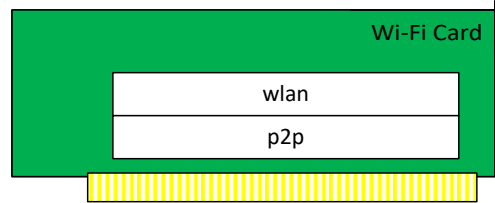


Figure 1-2 Virtual interfaces required for concurrent operation in Wi-Fi Direct

1.2.3.3 Group Formation and Device Addressing

Each Wi-Fi Direct equipped device can act as a GO or a GM. The selection of the GO depends on the group formation mode. According to the Wi-Fi Direct specification, there are three modes for creating Wi-Fi Direct groups, namely, standard, autonomous, and persistent [43]. In the standard mode, the devices involved in setting up the group negotiate among themselves to elect a GO. During the negotiation process, each device states its desire to become a GO by embedding an integer value called the GO intent in the “GO Negotiation Request” and “GO Negotiation Response frames”. This value ranges from 0 to 15, where a high value reflects increased interest in serving as a GO. The device with the highest intent value becomes the GO; a randomly selected tiebreaker bit is used in case of a tie. In the autonomous mode, one of the devices creates a group and declares itself as a GO; other devices connect to this group as GMs. Finally, in persistent mode, the devices save the information of the current Wi-Fi Direct group for future usage. If the same devices start a group again, the previous GO resumes ownership of the group.

Devices in Wi-Fi Direct groups have two addresses, the MAC address and the IP address. The MAC address is assigned by the OS to the device based on the MAC

address of the Wi-Fi chip. This address is used to identify the devices while creating the groups. The IP address is assigned to devices once they are attached to a group. A DHCP server running by the GO is responsible for assigning an IP address to each new device that joins its group. The IP address is used by the upper layer to identify and connect devices by using sockets.

1.2.3.4 Service Discovery in Wi-Fi Direct

Service discovery is a protocol that allows a device to explore what other devices offer before attempting to connect or communicate. This allows a device to define the scope of services that they support. For example, if three devices are in proximity to each other and two of them support chatting and the third supports media streaming, only the two with chatting capabilities should connect to each other.

Wi-Fi based WLANs support service discovery, meaning that devices in the same Wi-Fi network can define a set of offered services. If a device D_i is willing to search for a device that provides a certain service, D_i sends a service discovery request. Recipients of such a request respond back to D_i if they provide the solicited service. That way, D_i gets to know the IP addresses of the potential providers of the requested service so that it can initiate socket connections with them. We note that the devices in a Wi-Fi network must be associated with an access point to form a WLAN to be able to perform service discovery.

Likewise, Wi-Fi Direct also supports service discovery, but the concept is different. The devices do not need to be in the same LAN nor have prior connections to initiate a service discovery. The use of service discovery is optional, but if devices support it,

they become able to create groups based on certain services. The Service Discovery protocol follows the Generic Advertisement Service (GAS) protocol/frame exchange that is defined in IEEE P802.11u [44], where devices first announce their supported services by creating service records and storing them. Each of these records reports one of the features the device supports. For example, A device “X” that wants to connect to “Y”, first checks its supported features by sending a service discovery request. When device “Y” receives the request, it responds by sending its stored service records. If “X” and “Y” have matched services, they can proceed to form a group for data exchange. Thus, in Wi-Fi Direct, a device first performs the service discovery and then comes the association with other devices.

1.2.3.5 Android Implementation of Wi-Fi Direct and its Limitations

Android is one of the first operating systems that has implemented Wi-Fi Direct. Most Android devices starting from Ice Cream Sandwich v4.0 (API Level 14) are Wi-Fi Direct capable. Support for Wi-Fi Direct Service Discovery is added in Android since the JellyBeans v4.1 (API Level 16) release. However, the APIs for Wi-Fi Direct only provide basic support for connecting multiple peers in one P2P group. In general, Wi-Fi Direct implementation in Android has some technical limitations at both software and hardware level.

With the current Android APIs, a peer-to-peer system in the sense that every device can communicate with the others is not possible, since there are no APIs for informing devices in a group about the IP addresses of the other group members. Thus, a method of distributing IP addresses is required to allow devices to operate in a peer-to-peer

mode. Another problem is that although the android APIs make the IP address of the GO easily accessible to every device in the group, no API exists that allow a device to obtain the IP address of its own Wi-Fi Direct interface. The same applies when a device tries to retrieve its MAC. Thus, a way of finding these addresses is required first before attempting to distribute the list of IPs.

Starting from JellyBeans a device can simultaneously connect to a Wi-Fi Access Point and a Wi-Fi Direct group using the same Wi-Fi transceiver. It is not possible though for a device to connect to more than one Wi-Fi Direct group at the same time. This means that it is not possible to have certain scenarios like being a GM of two groups, being a GO of two group, or being a GM in one group and a GO in another group. One possible solution to overcome such limitation is to allow a device to be part of a group using its “p2p” interface and to connect to another group as a legacy device using its “WLAN” interface, given that it knows the credentials of the SAP of the other group.

Another implementation issue in Android is that the range of IP addresses assigned to a group falls in the 192.168.49.x/24 range, where the GO IP address is fixed at 192.168.49.1. Thus, even if there is a way to allow multi-group membership, the devices in different groups may not be able to reach each other due to IP address collision.

1.3 Research Goals

In emergency scenarios or natural disasters, the communication infrastructure may be damaged or severely degraded. In these cases, an alternative mean for sharing alerts must be established. Another application scenario is sharing data between a group of people are camping in a rural place that does not have any network coverage. In addition, in applications where devices are in the proximity of each other, such as in vehicular networks, they may suffer increase in latency, power consumption, and wireless service charges if they route their communicating through the telecommunication infrastructure. In these scenarios, it is desirable to enable data sharing in an easy and cost-effective manner that does not need the involvement of network infrastructure units.

We consider utilizing smart devices to form a network for sharing data that covers scenarios like the formerly mentioned ones. We choose Wi-Fi Direct as the underlying technology for enabling our work due to its distinct features, as explained earlier. Android is one of the popular mobile operating systems that have software support for Wi-Fi Direct. Thus, in this dissertation, we consider using Android smart devices for implementing our work.

1.4 Research Contribution

The main contribution of this dissertation is providing a data sharing solution through the development of a suite of protocols for infrastructure-less data sharing based on Wi-Fi Direct. The following highlights some of the features of the proposed protocols:

- Considering the case of sharing small chunks of data or alerts in a quick manner that is fast and is not limited by a group boundary, we have developed *ADS*. *ADS* uses service discovery in Wi-Fi Direct to exchange data between Android smart devices without requiring setting up any groups or having any prior connections. The data is stored locally using service discovery records. Other devices that are interested in such data use service discovery requests to obtain them. The approach also manages the forwarding of data and pruning old data.
- For a small set of users who need to exchange large amounts of data, we have developed *ELN*. *ELN* is a solution for data sharing between a set of smart devices that can be interconnected in one group. *ELN* provides group management capabilities to Wi-Fi Direct groups, a very important yet missing feature in Wi-Fi Direct. Through *ELN*, a group can dynamically adapt to topology changes such as addition or removal of peers.
- Finally, we consider the case of sharing data among large number of users that spans a wide area, which is larger than the boundary of a group. For such a case, we have developed *EMC* that dynamically creates Wi-Fi Direct groups of smart devices based on certain criteria, such as the battery specification of such devices. The approach then interconnects the formed groups using relay devices to achieve large scale data sharing. *EMC* utilizes the service discovery in Wi-Fi direct for distributing vital protocol specific data.

Other contributions can be summarized as follows:

- A. *Proposing a solution for assigning different subnet to groups:* To overcome the limitation of Android’s Wi-Fi Direct implementation that forces all the formed groups to share the same range of IP addresses, we have proposed ISNP. ISNP allows each group to have its distinct IP subnet, thus making it possible for groups to share data.
- B. *Validating the effectiveness of the proposed approaches through implementation:* We have evaluated the effectiveness of our approaches through implementation on Android devices and made an extensive analysis of the performance of such approaches. The results have confirmed the advantages of our protocols in terms of achieving timely data delivery, reduced overhead, and adaptation to topology changes.
- C. *Developing a simulator for Wi-Fi Direct:* To the best of our knowledge, there is no existing simulation environment for Wi-Fi Direct. We thus have developed a simulator based on OMNeT++ [83] along with the INET framework [84] which have several ready-made modules for the 802.11 protocol.

1.5 Organization

This chapter has presented the motivation and necessary preliminaries for P2P data sharing between smart devices. Chapter 2 discusses related work in the literature. Chapter 3 covers our alert dissemination protocol that uses service discovery in Wi-Fi Direct to distribute the alert data. Chapter 4 presents ELN, an efficient and lightweight protocol for connecting smart devices over a Wi-Fi Direct group. Chapter 5 presents

EMC for dynamically creating energy aware Wi-Fi Direct groups and interconnecting them using proxy members. Chapter 6 describes the ISNP protocol for IP subnet negotiation Protocol. Chapter 7 describes the extensions made to OMNet++ to support Wi-Fi Direct and to enable the implementation of our framework. The simulation results are also presented in Chapter 7. Chapter 8 concludes the dissertation with a summary of the contribution and outlines future research topics.

Chapter 2: Related Work

In this chapter, we discuss the published work, related to the contribution of this dissertation. Data sharing in the literature can be categorized based on the technology used. Since we utilize Wi-Fi Direct as the underlying P2P technology in this dissertation, we have designated a separate section for it. We have further categorized other P2P technologies them based on the type of the targeted network, such as vehicular networks, mobile networks, and ad-hoc networks.

2.1 Data Sharing using Wi-Fi Direct

Motta and Pasquale [45] were among the first to point out the potential of implementing mobile P2P systems using Wi-Fi Direct. They suggested applications that could benefit from Wi-Fi Direct such as text messaging, dissemination of traffic information, dissemination of emergency data, photo/video sharing during an event, and last-mile connectivity. The authors also explained how to use a new middleware for P2P based on JXTA, which employs distributed hash tables (DHT) to search for peers. They were aiming to implement the proposed middleware in Android once the Wi-Fi Direct API becomes publicly available. However, no progress has been reported in the literature on the implementation of such middleware. Nonetheless, such a study has motivated other researchers, e.g., [43][45], to explore the applicability of Wi-Fi Direct. Follow-up work has introduced protocols that allow the devices in a Wi-Fi Direct group to do true P2P data sharing [48][49]. The focus has then been shifted from intra-group to inter-group data sharing [53]. In addition, the use of the Wi-Fi Direct

service discovery protocol as a mean of data exchange has been explored in [50], [51], and [56]. A comparison with prior work is provided in the balance of this section.

2.1.1 The applicability of Wi-Fi Direct

Conti et al. [46] explored the possibility of creating opportunistic networks over Wi-Fi Direct by studying the latency in forming a group at the link layer. They experimented with different number of nexus devices, ranging from two to six. Their work is considered an extension to Camps-Mur et al. [43] who performed real experiments using only two devices. The experiments studied the performance in the standard, autonomous, and persistent modes of group formation. The results show that the group formation time can vary based on the timing of frames sent between the involved devices. They show also that attempting to connect to a group is greatly affected by the mode of the group formation. It was concluded that connecting to autonomous group is faster than connecting to a persistent group. Their explanation of such conclusion is that when the devices in the persistent group starts forming the group again, the members try to quickly connect to the group owner, which is still initializing the required interface and other parameter, and thus it may discard the request-to-join messages. After failure, the requesting members must wait for some time before attempting again. This work gave realistic metrics on how fast the devices could form and reform a group, as well as confirmed the suitability of Wi-Fi Direct for data sharing systems. Meanwhile, the authors of [47] studied the best topology for streaming multimedia contents between multiple devices. They conducted a performance

comparison between multiple strategies, like using access points, ad-hoc networks and Wi-Fi Direct and concluded that Wi-Fi Direct is the most suitable technology.

2.1.2 Intra-Group Data Sharing in Wi-Fi Direct

Peer management for iTrust over Wi-Fi Direct [48] is an attempt to port the iTrust protocol over SMS system to Wi-Fi Direct. iTrust is a peer-to-peer publication, search and retrieval system that enables peers to construct a mobile ad-hoc network for decentralized information sharing. A peer management protocol is proposed for adding new group members, where each peer opting to join a group should send to the GO a NEW_PEER message, which contains only the MAC address. The group owner can infer the IP address of the sender from the opened socket object. The group owner then compiles a list of MAC/IP address pairs for all peers and sends it to its members. However, no specific details were provided for how a peer departure from a group is handled. What is mentioned is that, if the connection is broken, iTrust performs automatic reconnection and rebuild the MAC/IP pairs. In addition, it is unclear why the service discovery feature in Wi-Fi Direct is not exploited to limit membership to only peers with capable services. This should have improved the efficiency of the group operation by defining contents as service types, and allowing a device that is willing to connect to a group to only look for devices with the required content type. In our work, we have considered the importance of service discovery in peer-to-peer. In addition, our work can adapt to topology changes.

Park et al. [49] proposed DirectSpace, a framework for collaboration between devices. The main goal is to provide a mean for sharing workspaces between users over Wi-Fi Direct. The framework is composed of two services, a connection service, and a collaboration service. The former handles the peer discovery and the connection/disconnection operations. The collaborative service is used for resource sharing and group management. To obtain a list of all peers in a group, the address resolution protocol (ARP) is used to translate MAC addresses into IP addresses. However, it is not clear how the group owner distributes this list to the members in the group. Moreover, the usage of ARP Tables is not a safe operation, as these tables are flushed periodically and not reliable. Also, DirectSpace did not benefit from the currently available service discovery API in Wi-Fi Direct. Doing this would relieve the network from associating peers that does not provide any required workspace to the group, which would improve the overall performance. In our work, we have addressed the service discovery part before connecting devices to the group. Unlike what the authors do in obtaining the IP/MAC addresses pairs, we avoided using the ARP tables and implemented ELN protocol for managing extraction and distribution of IP/MAC addresses.

Meanwhile, Chaki, et al. [51] proposed an approach for handling the group reformation issue in Wi-Fi Direct. It is known that when a GO disconnects from the group for any reason, the entire group breaks. Their approach tackles such a problem by choosing a list of Emergency GOs (EGO) whose responsibility is to restore intra-group connectivity. No consideration was given to inter-group issues. In [52] the authors

proposed an algorithm, called WD2, for automatically selecting group members based on the RSSI measurements. Thus, each device collects the RSSI reading from nearby devices and an Intent Value (IV) is calculated based on such collected measurements. The devices then exchange IV values. The device that has the best IV value creates the group. Although WD2 is validated on real devices, the scope of the work is limited to single group formation.

On the other hand, Wong et al. [56] exploited the service discovery feature of Wi-Fi Direct to distribute the credentials (SSID, Key) of the SAP created by a Wi-Fi Direct group to nearby devices in order to enable them to connect to such SAP. Their goal is to create a mesh router using a Wi-Fi Direct group. During the implementation, they found out that connecting a device to the group must be manually confirmed; however, connecting the same device as a legacy client to the SAP of the group does not need confirmation. Such finding has allowed them to bypass the confirmation process. However, they did not exploit service discovery to share actual data between devices other than the SSID and the key, unlike our ADS protocol that has such ability feature. In addition, they did not attempt to cover multiple groups. Likewise, Menegato, et al. [50] utilized the service discovery protocol in Wi-Fi Direct for exchanging certain information among devices to aid them in creating clusters (groups) for data sharing. Three algorithms were proposed for selecting cluster heads. Nonetheless, no implementation on actual devices was attempted. In addition, their work is limited to a single group formation without consideration of inter-group communication.

2.1.3 Inter-Group Data Sharing:

Duan et al. [53] were the first to propose a method for establishing multi-group communication in Wi-Fi Direct. They base their work on the fact that concurrent operations are allowed in the Wi-Fi transceiver using both the “WLAN” and the “p2p” interfaces in the same time, i.e., a device can be connected to a Wi-Fi Direct group and a Wi-Fi network in the same time. Wi-Fi Direct supports legacy devices by letting the GO create a software access point that legacy devices can associate with. Thus, they connected two groups by letting the GO from the first group to connect as a legacy client in the second group using the “WLAN” interface. The authors then experimented with three groups and showed how to connect them together. A problem they faced was that all groups were sharing the same subnet (192.168.49.x/24). To overcome this, they used a combination of unicast and multicast communications. Although they have investigated inter-group interaction in Wi-Fi Direct, they did not propose a dynamic way to create the groups and automatically connect them. What they used for validating their work was a manually created topology. In addition, their choice for having the GOs to perform the legacy connection imposes constraints on how the GOs should be far from each other and the number of groups that a certain group could reach directly.

Laha, et al. [54] proposed a method for clustering several Wi-Fi Direct devices into multiple groups based on LEACH [55]. The idea is to allow the devices to rotate the GO responsibility and efficiently manage the consumption of their batteries. The created clusters, or groups, would then exchange data, or pass it to a base station. The authors validated such a scheme through simulation without considering the challenge

associated with implementation on actual Wi-Fi Direct devices. In addition, no solution was proposed to overcome the IP subnet limitation if their work is to be implemented on real devices.

2.2 Data Sharing using other Technologies

Supporting efficient data sharing has been investigated in the context of multiple applications, namely distribution of alerts, warnings, or general information in VANET [57]-[62], in networked mobile devices [63]-[66], and in first responders ad-hoc networks [67]-[74]. Unlike published schemes, our work is not tied to a certain application; it can be used in vehicular networks to improve road safety or in first responder application to enable efficient handling of disasters, etc. In addition, some of the published protocols require an existing connection or special hardware, which is not available in most cases.

2.2.1 Data Sharing in VANETs

Doukha et al. [57] proposed a protocol for disseminating urgent alerts in VANET. The idea is to combine the use of unicast and broadcast transmissions to achieve network-wide distribution task with low latency. A message is sent twice, one using unicast to the farthest node from the sender and again using broadcast to reach nearby nodes. The node that receives the unicast message then retransmits it again using the same method. This way they limit the number of broadcasts, as only certain nodes are responsible of sending broadcast messages. Suriyapaiboonwattana et al. [59] also considered reducing alert delivery latency as a design objective for their protocol. Unlike [57], they opt to improve the rate of successful alert message delivery.

Basically, conventional broadcasting is enhanced by an adaptive adjustment to the wait-time until retransmission is attempted. The number of duplicated message that are received during the wait-time controls the next setting.

Another adaptive scheme for alert dissemination in VANET is presented in [60]. The idea is to gather the information about the characteristics of the road in the area using street maps and estimate the density of vehicles based on beacon frames. Both parameters are then used to improve the message broadcasting process. Meanwhile, the authors of [58] proposed two protocols for distributing alerts. The first deals with the selection of the most suitable node to act as a relay for broadcasting alert. The second protocol addresses the case of multiple hazards occurring at the same time in the road. In such a case, the protocol finds the most effective vehicle to initiate the alert message while taking into account the latency, interference, and reliability. In both protocols hello message are used to gather the information about neighbors.

In [61], a protocol for disseminating local advertisements is presented. The authors are interested in announcing local services on the road such as gas station, repair shops, restaurants, etc. to passing vehicles. They pursue a push model and a distance based forwarding for announcing these services. The objective is to reduce the number of required transmission to cover a given area and to limit the medium access collisions. To achieve such an objective, a node that received a message waits for a short period to allow for other copies of the same message to arrive. Doing so allows the node to know if it is the furthest from the original sender of the message. This approach favors

directing a distant relay to forward data from the previous sender in order to cover as much area as possible with the fewest number of broadcasts.

Other work, e.g. [62], proposed certain modification for the 802.11p MAC protocol to allow efficient dissemination of emergency messages. The modifications include muting the back-off procedure in case of sending emergency messages, and introducing a separate queue for holding such type of messages. Although the proposed changes enable faster transmission in case of emergency, disabling the back-off algorithm means more collision on the channel. Thus, such protocol is suitable for simple scenarios involving few nodes.

2.2.2 Data Sharing in Mobile Devices/Networks

Huang et al. [63] benchmarked a commercial network, AT&T Enterprise Messaging Network (EMN), to assess its ability to disseminate alerts. They built an Alert Dissemination engine (ADE) as a service on top of EMN. These benchmarks helped in determining the scalability parameters of the system for a given hardware configuration. They also experimented with splitting the alerts into smaller messages and showed that splitting is good for decreasing the latency in disseminating alerts. They also found that excessive splitting is not beneficial, so the number of message segments must be kept small. It is worth noting that the scope of their work and the results are tailored for a special network setup.

An attempt to use smart phones for creating an emergency communication service was introduced in [64]. The authors proposed attaching of an FM transmitter module to the phone. This module can then be utilized by an application to send an SOS

message encoded by Morse code to emergency responders. A message could be forwarded by several users until it reaches the intended responders. This approach is not suitable for every user though, as it requires a special module to be attached to the device.

In [66] Teranishi and Shimojo proposed a system for disseminating social network messages such as Twitter in case of losing the connection with the cloud. They used a P2P based overlay to store and forward the messages. A device waits until another device comes within its range and sends the messages to it. This is repeated until the cloud is reached and the message is delivered to the social network. Although the approach is useful for sending SOS messages via twitter in case of disasters, the alert may not be of a much benefit, since other interested parties may not be able to reach the cloud too.

2.2.3 Data Sharing in Ad-hoc Networks

In [67], Thomas et al. discussed the implementation of their Smart Phone Ad-Hoc Networks (SPAN) protocol on Android. Their goal was to allow smart phones to create mesh networks. They also promoted the concept of off-grid communications, where peers could talk to each other without the need for a cellular connection. Since forming an ad-hoc network is not supported by Android, the command line utility “iwconfig” was used to configure the connection. Because not all Android devices support “iwconfig”, the kernel was modified for some of these devices to allow the usage of the command. Nonetheless, this approach for P2P networking is not suitable for contemporary users, as it requires the device to be at least rooted to use the “iwconfig”

command, which is not a trivial job for average users to do. Such a restriction makes this method impractical. Our work, on the other hand, does not require any modification to existing systems as we use Wi-Fi Direct, which is supported by the current Android APIs. Although we had to overcome some limitations in the Android API, no rooting is required. Thus, any application that utilizes our framework could work correctly in stock version of Android.

Kolios et al.[68] introduced a new paradigm for alert dissemination called EnE. This paradigm is specially designed for emergency ad-hoc networks. EnE relies on specific link layer protocol called LC designed by the authors. The LC protocol extracts the network topology then an LTE-Direct [75] feature in the protocol is used to locate nearby neighbors.

Most of the work done in the context of data sharing in ad-hoc networks assumes that certain devices are available to do the job. In sudden event, e.g., an earthquake, the unavailability of devices in the affected area may delay the rescue operation. Our work in contrast allows data to be shared without the need for having any special devices. We utilize the smart portable devices that most of people carry in their pockets, which allows faster notification and higher responsiveness to urgent situations.

Chapter 3: Alert Dissemination Protocol Using Service Discovery in Wi-Fi Direct

In this chapter, we propose a protocol for alert dissemination among smart devices using service discovery (ADS) in Wi-Fi Direct [79]. ADS relies on the service discovery feature of Wi-Fi Direct for distributing alerts or short messages to nearby devices without having any prior connections and thus avoiding the setup delay in creating Wi-Fi Direct groups. As ADS does not need any infrastructure, connections, or groups for data exchange, it is suitable for scenarios where reducing the latency is the most important aspect, such as emergencies. When a device D_i needs to send an alert, it creates a service announcement record and replaces the service description with the alert data. When D_i receives a request from another device D_j to list its services (using a service discovery request), D_i sends the stored record which contains the alert data. Nearby devices, which continuously probe for services by sending discovery requests, receive this announcement and store the record. The alert stored in the record is forwarded to other devices using the same mechanism. In addition, we present a mechanism for discarding inactive alerts. ADS also avoids the flooding of the network by forwarded records. ADS is validated by implementing a hazard propagation application for Android. The performance of the protocol in reporting alerts and pruning unneeded ones is also analyzed.

3.1 Problem Statement and Solution Strategy

An efficient and fast alert dissemination protocols is required in case of emergencies. For example, the vehicles on the road may alert each other about accidents, dangerous

pavement cracks, traffic congestion, etc. To implement such a protocol, two methodologies may be pursued. The first is to proactively send alerts as soon as a device detects an event or forward it once a notification from another device is received. The second option is to wait until receiving an inquiry or update request (i.e. publish/subscribe model). Obviously, the latter approach is more optimized as it allows only the interested devices to receive the alert messages. Such an approach aligns with service discovery concept in Wi-Fi Direct.

ADS exploits the service discovery in Wi-Fi Direct to support timely alert dissemination in a publish/subscribe manner. The Wi-Fi Direct service discovery frames are utilized to encapsulate the alert details. Thus, ADS requires no infrastructure or known topology. The devices themselves can be mobile or stationary. The advantages of this approach are: 1) it allows optimized delivery of alerts; 2) it does not require any additional hardware; 3) it can be implemented on a large scale without extra cost. An example usage of ADS is shown in Figure 3-1, where a hazard information been disseminated to four smartphones. When a device detects something unusual, it records an alert information in a service discovery record. Interested devices ask for existing alerts by sending a service discovery request. A nearby device with stored alert records responds by encapsulating the details of each alert details in a separate service response frame. The ADS protocol is explained in detail in the next section.

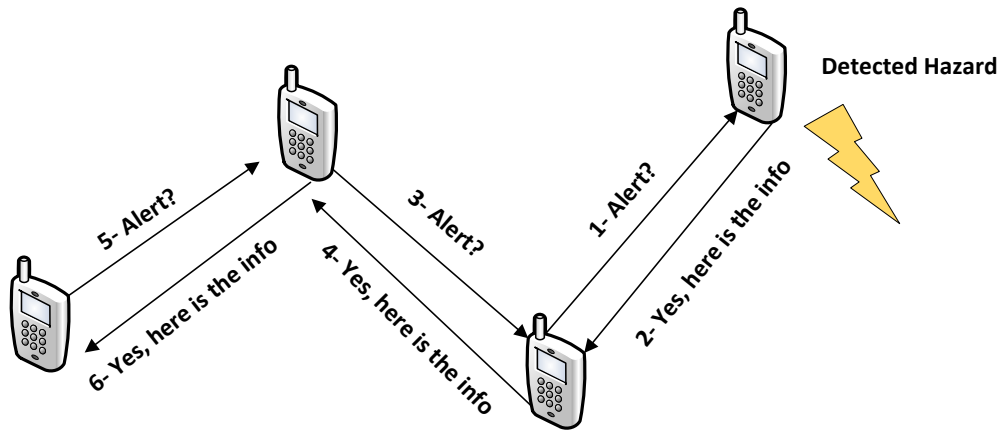


Figure 3-1 Example usage of ADS.

3.2 ADS Protocol

Our ADS protocol for alert dissemination using the service discovery of Wi-Fi Direct is composed of two parts for: (1) managing local alerts, and (2) managing remote alerts. As the name indicates, the distinction between local and remote alerts is based on which device generated such an alert. A locally generated alert reflects an event that the device has detected; thus, the device can track such an event and maintain its own record. In addition, the local alert manager is responsible for storing alerts as service discovery records and responding to inquiries from nearby devices. The remote alert manager handles the dissemination of notifications about events detected by other devices. The protocol itself does not need the devices to be connected in any way. The only requirement is that a device applying ADS should have a Wi-Fi transceiver that supports Wi-Fi Direct and an operating system with APIs that supports the Wi-Fi Direct standard and its service discovery protocol. The details of ADS are provided in the balance of this section.

3.2.1 Service Record

The ADS protocol starts by registering a common instance name for the devices that are willing to use the protocol to receive alert. The instance name is part of the Wi-Fi Direct service discovery protocol. Other devices in the vicinity that are not interested in running the ADS protocol will not be engaged, as their instance name will be different. A service discovery record is used to store each alert that a device knows about so far. These records are exchanged with other devices for the sake of alert dissemination. The format of the service discovery record for ADS is shown in Figure 3-2. The deviceID is a unique identifier for the device, which can be the MAC address. The seqNo is an auto-incremented number that is used to differentiate between subsequent records. The data field in the record stores the alert information; for example, in hazard detection scenarios, such a data field includes the location and type of the hazard. The isValid field is a Boolean indicator for whether the alert is still valid (in effect). Setting this field to zero means instructing nearby devices to dismiss any existing copy of this alert record.

ADS Record				
SERVICE DSICOVERY HEADER	deviceID	seqNo	data	isValid
	Unique Id for the device	Unique idetifier for the alert	Alert details (e.g., Hazard location)	Indicates whether the alert is to be discarded

Figure 3-2 The format of the alert record stored on the device.

ADS maintains two internal tables to store local and remote alert entries. An alert entry contains an additional time-to-live (TTL) field that is used to prune inactive alerts

once these values reach zero. The TTL value is initialized to A_{TTL} , which is determined by the applications, e.g., typical time until an accident is cleared from the road, etc. For local alerts, the TTL is gradually decreased, e.g., every T_{decTTL} seconds, once the event ends and the `isValid` field in the corresponding record is set to false. The gradual decrease of TTL, rather than the immediate removal of the event record, is to allow other inquiring devices to update their alert entries. Meanwhile, the TTL of a remote alert is decreased on every T_{decTTL} seconds with $T_{decTTL} < A_{TTL}$, and is reset again to A_{TTL} if the same alert is received again. An additional field, called `srcDevAddress`, in each entry in the remote alert table is provided to identify the device that sent the alert. That field is different from the `deviceId` field, which notes the originator of the alert. If an alert is to be forwarded from device to another, the `srcDevAddress` is changed to reflect the last device that sent the alert, while the `deviceId` stays the same. The `srcDevAddress` is the MAC address of the device, which can be extracted from the service discovery frame of Wi-Fi Direct.

A detailed flowchart for the ADS protocol is shown in Figure 3-3. The management of both local and remote alerts is discussed in the next sections.

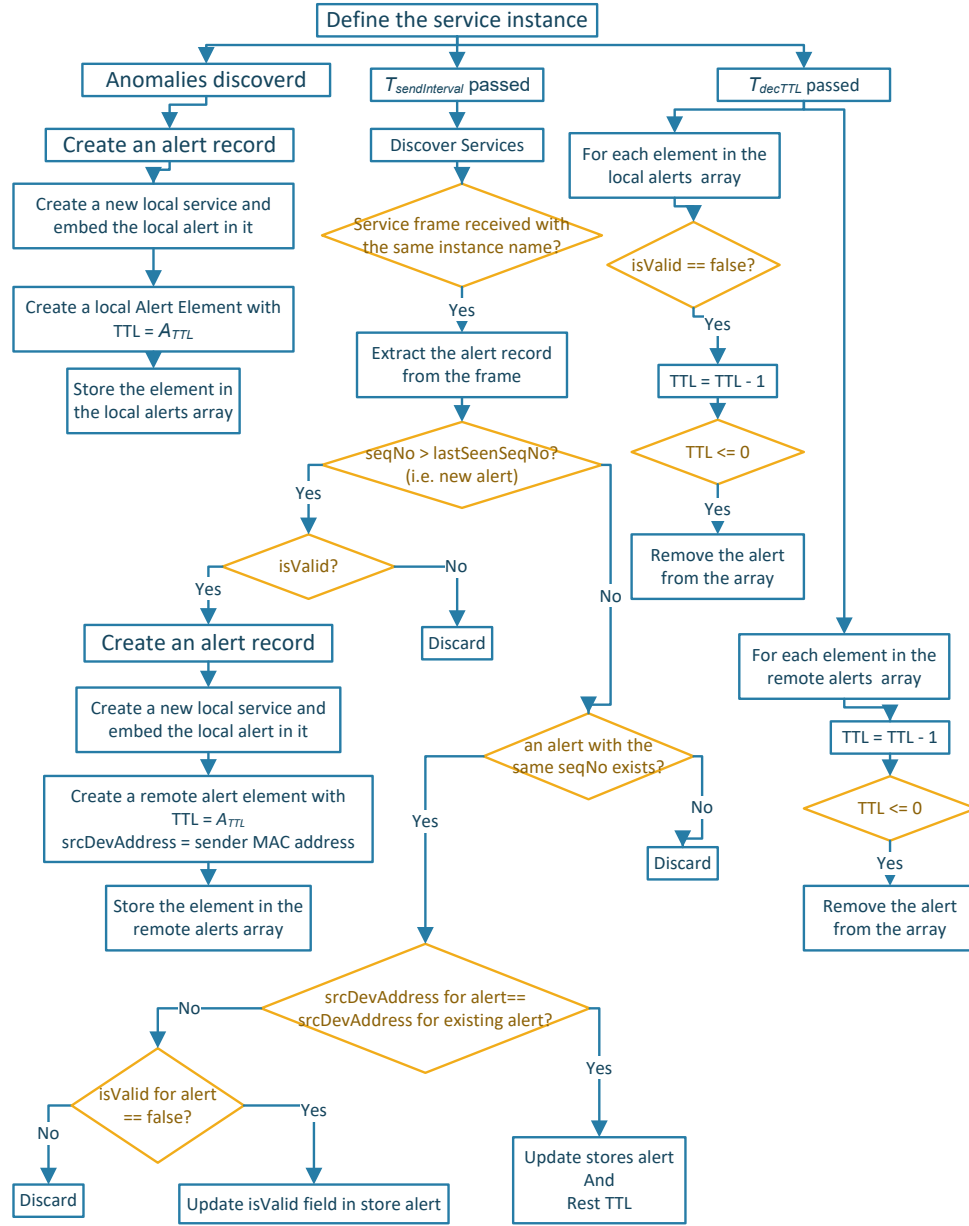


Figure 3-3 A flow chart description of the ADS protocol.

3.2.2 Local Alert Management

Upon detecting a noteworthy event, an alert entry is created and populated with the unique ID of the device, the current sequence number, and the alert details. The record is set to valid (isValid field is set to true) with a TTL value of A_{TTL} . Such an alert entry

is then stored in the local alert table. Additionally, a service discovery record that has the format shown in Figure 3-2 is created based on such alert element and stored as a local service that the device would support. A device, which runs ADS protocol and is within the Wi-Fi range, sends a service discovery request to learn about current alerts in the area. Upon receiving such a request, stored service discovery frames will be sent containing alert records along with other information that is part of the service discovery frame (such as MAC address of the device sending the frame).

Once the event becomes inactive, the corresponding local alert entry is invalidated, i.e., the `isValid` field is set to false, in the service discovery record of that alert. Since the alert is still stored in the device, it will be sent to any querying device to inform it that this event is not active any more. This way other devices know about the deactivation of the alert by noticing the `isValid` field, so that they can inform others as well. After invalidating a local alert, the local management part of the protocol starts to decrease the TTL value for such alert every T_{decTTL} seconds. Once the TTL value reaches zero, the alert is removed.

3.2.3 Managing Remote Alerts

A device applying the ADS protocol gets informed about events detected in the area by sending service discovery requests every $T_{sendInterval}$ seconds. Like T_{decTTL} and A_{TTL} , the value of $T_{sendInterval}$ depends on the application and on how dynamic the environment is. A device that is in range will react to the request by sending a service discovery response indicating its supported services in the context of Wi-Fi Direct, which reflect the stored alerts. As mentioned before, we modified the records stored in the service

discovery frames to include alerts information. For a received service discovery frame, ADS extracts the alert record and processes it. First, the alert is checked by looking for previously received seqNo from that deviceID. A hash-table that contains the last seen seqNo from each deviceID helps in performing such check. A received alert is deemed new if its seqNo is larger than the last seen seqNo from the same sender.

For a new alert, the isValid field is inspected. In case the alert is invalid, the message is discarded. After confirming its validity, an entry in the remote alert table is created. The entry includes the data extracted from the alert record plus a TTL value set to A_{TTL} and a srcDevAddress field populated with the MAC address of the device that sent the service discovery frame. ADS forwards the received alert to let other devices, which may not have received this alert or are not in the range of the sender. The forwarding is done in a similar manner to local alerts, i.e. by declaring a locally supported service that contains the received remote alert record. The device then responds to service discovery requests from other devices by announcing the locally supported services, which contains both local and remote alert records.

If the received alert is not new, i.e., seqNo is less than or equal to the last seen seqNo from the sender, it means that this alert is an update to a previously received alert or it is a dangling alert that is no longer valid. First, ADS checks remote alert table using the seqNo. If no match is found, the alert message is discarded. Otherwise, the srcDevAddress field of the stored alert is checked to see if it matches the MAC address of the sender. We have two cases here: (1) the sender of the alert is the same device that sent the previous alert, or (2) the sender of the alert is different from the previous

sender. In first case, ADS updates the information of the stored alert, such as the alert details and the isValid field. If the alert is still valid, the TTL value is reset to A_{TTL} . In the second case, we do not update any information regarding the alert except in a special case for the isValid field. If the isValid field of the received alert is false and the value for stored one is true, the isValid value for the stored record is updated to speed up the pruning of such inactive alert.

As in the local alerts case, the ADS protocol prunes inactive remote alerts. Each T_{decTTL} seconds, the TTL value of all remote alerts is reduced by one. As indicated previously, the TTL value could be reset to A_{TTL} again if an updated record is received. Once the TTL value reaches zero, the remote hazard is removed completely from the remote alert table and the declared locally supported service is removed.

3.3 ADS Implementation

To validate the features and implementation of ADS, we have developed an Android application for distributing alerts regarding detected hazards. A version of this application is available at GitHub¹. The implementation of our protocol allowed smart devices to disseminate hazard alerts between each other without the need for a connection or any infrastructure. This application is written in AndroidStudio using Android SDK. The application targets Android API level 16 to be able to use the Wi-Fi Direct service discovery APIs. The application is composed of the following

¹ <https://github.com/ashahin1/WiFiDirectServiceDiscoveryTransfer>

components Main Activity, and Hazards Class. Table 3-1 shows a brief description for what each component does.

Table 3-1 Application components and their description

Component	Description
Main Activity	<ul style="list-style-type: none"> - Performs the service discovery initializations. - Creates a google maps object to display both local and remote alerts (hazards) on it. - Enables declaring local alerts (hazards) in by tapping on the map. - Enables voiding local alerts by tapping on them on the map. - Stores local alerts in service discovery record. - Responds to service discovery requests by sending the stored alert records. - Periodically decreases the TTL values for inactive local and remote alerts. - Periodically prunes alerts with TTL values less than or equal to zero.
Hazards Class	<ul style="list-style-type: none"> - Declares a hazardItem object that is composed of the srcDeviceAdress, deviceID, seqNo, location, isValid, and TTL fields. - Manipulates arrays of local and remote hazardItems by adding removing and updating such items.

The application user can tap on a displayed map to declare a local alert or hazard (places a red marker on the map). The location of tapping on the map is used to declare the GPS coordinates of the alert. Other users nearby receive an indication of the hazard on their maps (blue markers on the map). We used four Android devices (two Nexus 4 phones and two Samsung Galaxy Tab 2 7.0 tablets) to test the application. In this test, we fixed T_{decTTL} , $T_{sendInterval}$, and A_{TTL} to 1, 5, and 30 respectively. At the beginning, we started the application in all devices. We tapped on each device's map to create local alerts (Hazards). We then confirmed that the devices started to exchange their local alerts. Because of such successful exchange, the devices displayed the remote hazards they received on their maps with blue markers. Finally, we removed all local hazards on all devices and the remote hazards were removed from the other devices. Figure 3-4

shows two screenshots from the two devices while exchanging alerts. As we can see, one of the devices has two local alerts marked red and the other has only one local alert. Both devices successfully exchanged the alerts data, as we can see blue markers on both devices indicating remote alert.

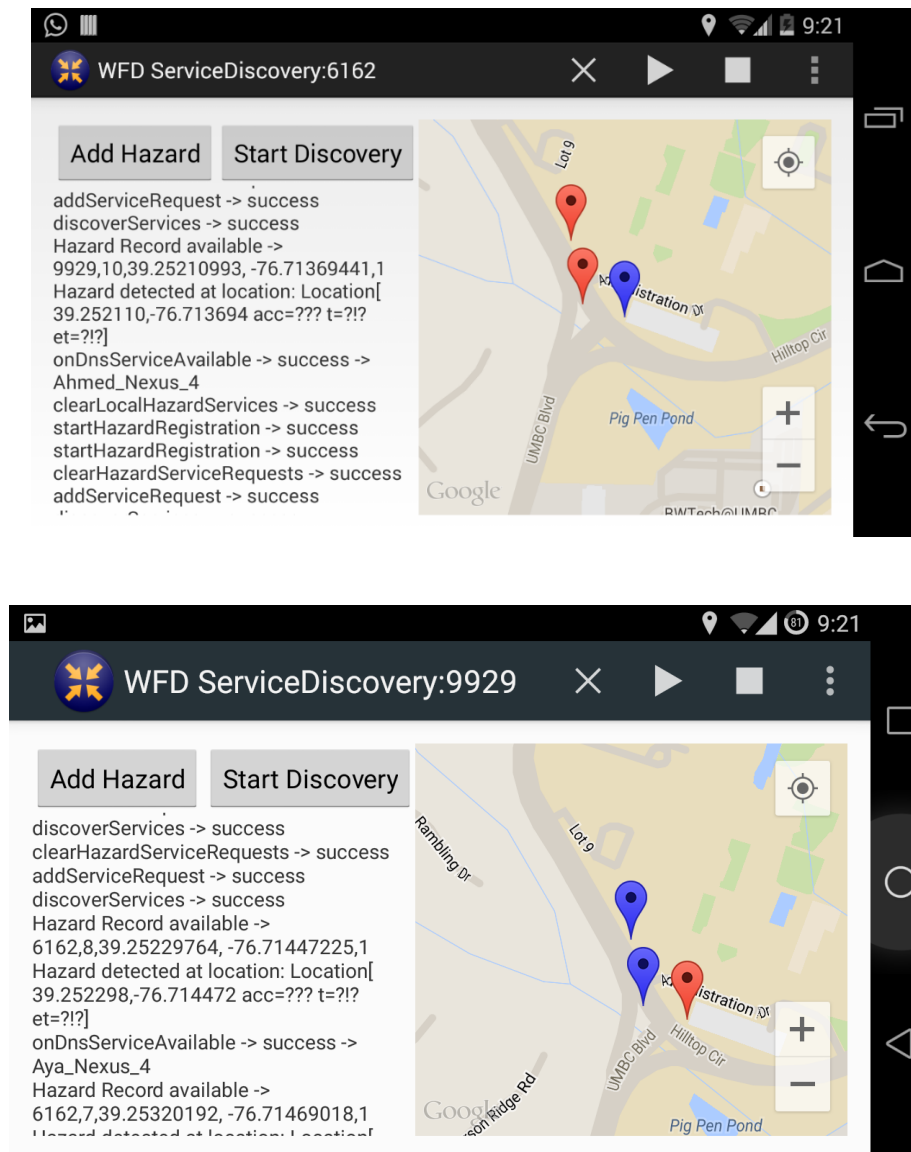


Figure 3-4 Screenshots from two devices showing local alerts (marked red) and remote alert (marked blue) from the other device.

3.4 Performance Analysis

In this section, we analyze the performance of the proposed ADS protocol. The goal of the evaluation is to capture how fast the reporting of a new alert and the pruning of an existing alert are. We assume in this analysis a Wi-Fi transceiver that supports the 802.11n standard. Typical transmission ranges for such a transceiver are 70m for indoor and 250m for outdoor. For the sake of this evaluation, we assume a transmission range of 120m. The transmission speed for 802.11n ranges from 54 Mbps to 600 Mbps. We choose 54 Mbps, as it is the most common data rate. We also assume that the service discovery request and the service discovery response frames to have the same length of L bytes.

3.4.1 Alert Reporting

Figure 3-5 shows the actions taken to report an alert. In the figure, device X is querying a nearby device, Y , about its local alerts by sending a service discovery request. The request should reach Y after T seconds, where T is the time needed to deliver a service discovery frame. At that time, device X has nothing to report so it did not send any response. Just after Δ seconds from receiving the request, device Y detects an alert and stores it. In ADS, a device discovers alerts every $T_{sendInterval}$ seconds. Thus, after $T_{sendInterval}$ seconds are elapsed from the first request, device X tries again. Device Y at that time reports the alert back to X . After T more seconds, Device X receives the alert. Therefore, an alert could take $T_{sendInterval} + 2T - (T + \Delta) = T_{sendInterval} + T - \Delta$ seconds before being available to nearby devices.

<u>Time (sec)</u>	<u>Device X</u>	<u>Device Y</u>
0	Send service discovery request	
T		Received the request but nothing to report
$T + \Delta$		Alert found
$T_{sendInterval}$	Send service discovery request	
$T_{sendInterval} + T$		Received the request and replied instantly
$T_{sendInterval} + 2T$	Received the response	

Figure 3-5 A graph showing the actions taking to report an alert.

The time T is composed of the propagation delay, T_p , and the transmission time, T_t . For a wireless medium, T_p depends on the distance between the two devices. Assuming that device X and Y are separated by a distance equals to the maximum range for Wi-Fi,

$$T_p = \frac{120}{\text{speed of light}} = \frac{120}{3 \times 10^8} = 0.4 \mu s$$

T_t depends on the transmission rate, the length of the frame, thus

$$T_t = \frac{L}{54 \text{ Mbps}}.$$

For $L = 5000$ byte, we will have $T_t = \frac{5000 \times 8}{54000000} = 0.7 \text{ ms}$. Therefore

$$T = 0.4 \mu s + 0.7 \text{ ms} \cong 0.7 \text{ ms}.$$

The range for Δ is $0 < \Delta \leq T_{sendInterval}$. Thus, an alert could take any value between $T_{sendInterval} + 0.7 \text{ ms}$, 0.7 ms to be noticed by other devices. For a rectangular area, whose diagonal is N multiples of the Wi-Fi range, to be covered by alerts, we need a time between $N(T_{sendInterval} + 0.7) \text{ ms}$ and $N \times 0.7 \text{ ms}$ to reach the farthest device. From that, we conclude that reducing the $T_{sendInterval}$ value helps in

speeding up the reporting of alerts, however it cannot go below T . For very small values of $T_{sendInterval}$, the network would be overwhelmed with request and response packets. That means the setting of $T_{sendInterval}$ should be selected based on the anticipated number of devices that are in the same Wi-Fi range.

3.4.2 Alert Pruning

The ADS protocol performs the pruning of inactive alerts with the help of the `isValid` field and the TTL value for a given alert. For local alerts, an inactive alert is determined by having a false value for the `isValid` field. The TTL value for an inactive local alert is decreased every T_{decTTL} seconds until it reaches zero. At that time, such an alert is pruned. As the TTL value is initiated to A_{TTL} , it takes $A_{TTL} \times T_{decTTL}$ seconds from the deactivation time to prune a local alert.

For remote alerts, the TTL values are decreased every T_{decTTL} seconds whether it is active or inactive. The TTL value is reset back to A_{TTL} if the alert is received again from the same sender. Thus, a remote alert is pruned after $A_{TTL} \times T_{decTTL}$ seconds from the last time it is seen. Any alert deactivated between two service discovery periods takes the same $A_{TTL} \times T_{decTTL}$ seconds to be pruned. The value of T_{decTTL} should be chosen to be smaller than $T_{sendInterval}$ to speed up the pruning of inactive alerts. For A_{TTL} , we should choose its value such that $A_{TTL} \times T_{decTTL}$ is larger than $T_{sendInterval}$ to avoid pruning remote alerts that are still active.

3.5 Conclusions

In this chapter, we have presented ADS, a protocol for disseminating alerts or short messages between smart devices. The protocol uses the service discovery protocol in Wi-Fi Direct to perform the data distribution without the need for infrastructure or any existing connections between devices. The main components of the protocol are the local alerts management module and the remote alerts management module. The former generates local alerts, stores them, and distributes them when requested. The remote alerts module is responsible for querying for alerts on other devices, storing them, and redistributing these alerts to other devices. ADS also avoids flooding the network with many alerts, removes duplicate alerts, and prunes inactive alerts.

An application is implemented to validate the ADS protocol using level 16 of the Android APIs. The implementation has confirmed the applicability of the ADS protocol through testing on four smart devices. The performance of ADS is analyzed for reporting new alerts and pruning inactive ones. The analysis has provided guidelines to how to choose the ADS parameters to achieve low-latency reporting and update of alerts. We developed a unified simulator for all the protocols in this dissertation, which is presented in chapter 7. The performance of ADS in many different scenarios has been studied and reported by the help of such simulator, as shown that chapter.

Chapter 4: Efficient P2P Networking of Smart Devices over Wi-Fi Direct

In this chapter, we present an efficient and lightweight protocol for peer-to-peer networking of Android smart devices over Wi-Fi Direct (ELN) [76], a protocol for allowing a small number of users to share large amounts of data. The protocol includes a connection setup phase and a group management phase. The former enables different devices with the same interests to form a Wi-Fi Direct group. The group management phase provides the necessary means for handling dynamic group membership and adapting to topology changes to allow group members to communicate with each other in a peer-to-peer fashion. The ability of Wi-Fi Direct to preform fast data streaming (which is a required feature for peer-to-peer systems) is validated by writing an application that streams the accelerometer readings between two devices. ELN is implemented by building a group chatting application on four Android devices. We also monitored how ELN responds to different connection/disconnection scenarios. The overhead imposed by our protocol is also calculated.

4.1 ELN Approach

ELN is composed of a connection establishment phase and a top-layer group management phase. The connection establishment phase allows only the devices with the same interest to connect with each other. The group management phase allows treating the Wi-Fi Direct topology, which is by convention a star network, as a mesh network. ELN does so by providing a mean of distributing the peers' IP addresses,

facilitating transport layer connections and managing addition and removal of peers from the group. ELN is described in detail in the balance of this section.

4.1.1 Connection Establishment Phase

The goal of the connection establishment phase is to allow the devices to define their supported service types, and to filter nearby devices based on that. Thus, this phase allows only the devices that provide the same services to connect. Before attempting to form any Wi-Fi Direct group or connecting to an existing group, a device has to announce its supported services. The Android APIs for Wi-Fi service discovery has an option to include a service record along with the service type that the device can provide. The service record is used for exchanging additional data, which is used in the connection establishment.

In this phase, each device adds a uniqueID and the availability status to the record. The uniqueID is calculated once by the device and is stored for future connections. This ID is formed by the concatenation of two randomly computed numbers. The first number is an integer random number and the last is a float random number. The probability of having a completely unique number is very high, as the devices generating these random numbers are not tied to the same clock and also they are not running at the same exact instance of time. This uniqueID is used to differentiate between devices in case they have the same name. It is also used by the next phase for retrieving peers records. The availability status allows connecting devices to delay or postpone the connection to a device until that device is available. An optional username can be included in the service record. When included, this username will be

used by peers as a more expressive identifier. In case of omitting the user name field, the device name is used.

Upon receiving a service discovery announcement from a peer, the device retrieves the information in it and then checks for the service type that such a peer provides. If both of this device and the connecting peer agree on the same service type, the device continues processing the information, otherwise the announcement is discarded. The service record is then retrieved and stored. If a username is included the device uses it when displaying the found peer to the user, otherwise the peer's device name is used. The availability status allows the user to know when to attempt the connection with the device. Flowcharts for the connection establishment phase are shown in Figure 4-1 and Figure 4-2.

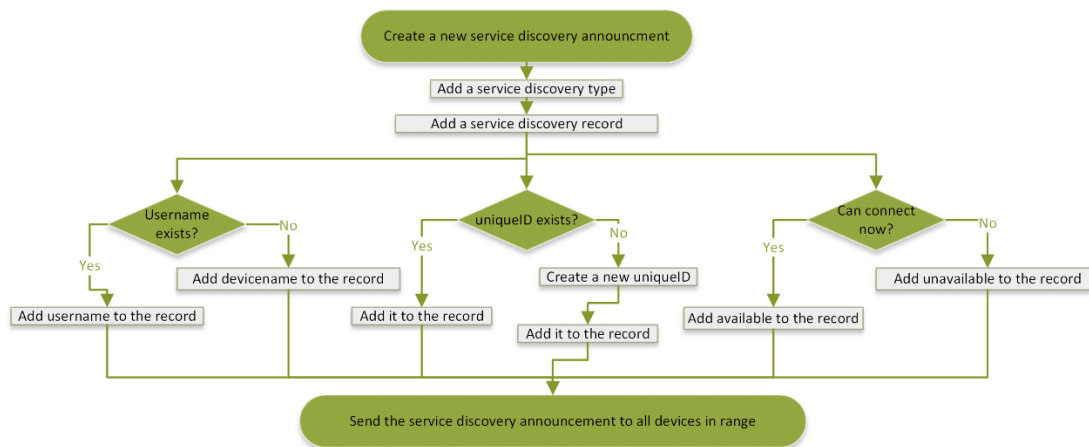


Figure 4-1 The connection establishment phase from the sender side.

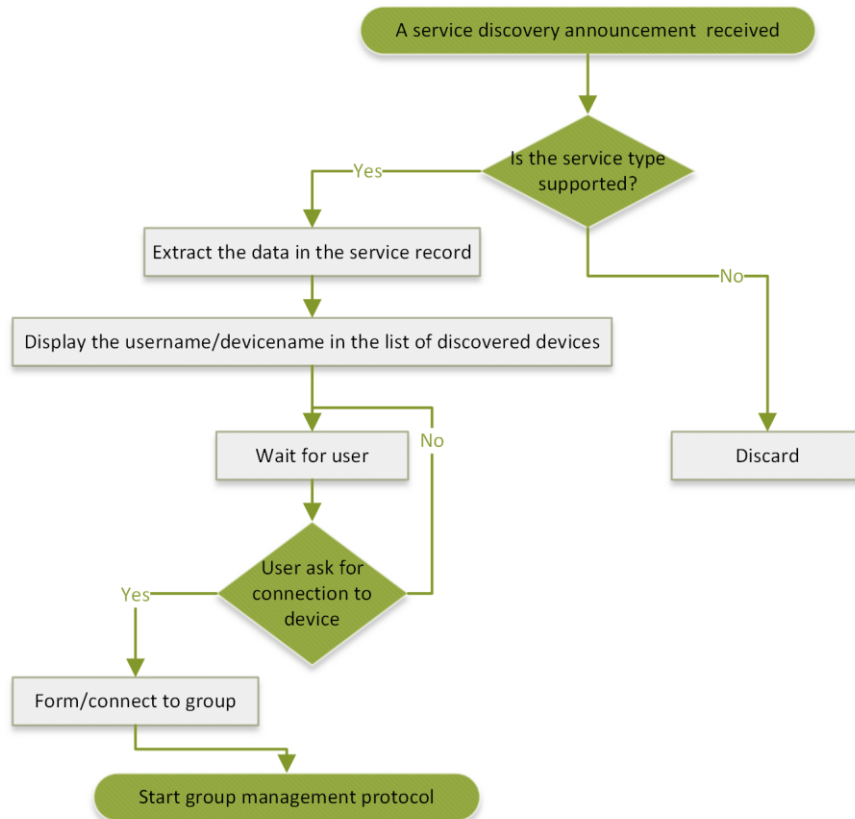


Figure 4-2 The connnection establishment phase from the receiver side.

4.1.2 Group Management Phase

Once a group is formed based on service matches, the group management phase forms peer-to-peer links among the group members at the level of the transport layer. As part of the Wi-Fi Direct connection setup, an embedded DHCP server that runs in the group owner assigns the IP addresses for the group members. Members in a group may not know about other member in that group, thus the group management phase in ELN distributes the information of the GMs to everyone in the group. We will explain the operation of the proposed management phase in the balance of this section.

4.1.2.1 Layers of Connections

The group management phase uses two layers of connections one for management purposes and the other for data exchange purposes.

Management Connections: As the name indicates, the management connections are for managing the group. The group owner receives peers' information and sends the list of peers to all members. Upon creating the Wi-Fi Direct group, the GO opens a server socket and binds it to a predefined management port. As the GO IP address is already known to all GMs as part of the Wi-Fi Direct APIs, every GM connects to the server socket of the GO. The final topology of the connections at the management level is a star topology that originates at the GO as shown in Figure 4-3.

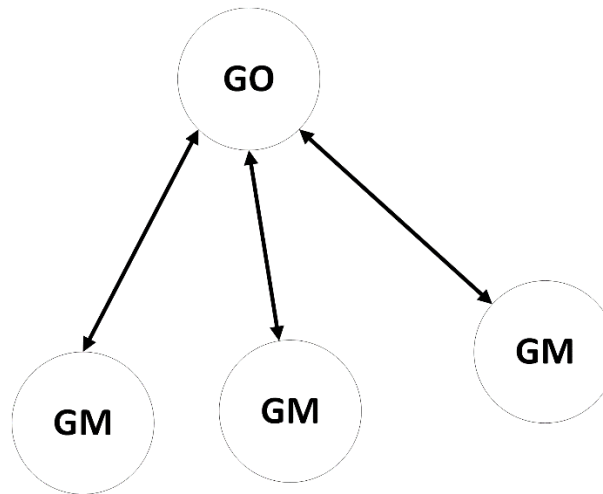


Figure 4-3 The topology for the management connections.

Data Exchange Connections: The GMs need to obtain a list of other peers' information from the GO for opening data exchange connections with each other. Meanwhile, every member in the group opens a server socket and binds it to a known data-exchange - port. The server socket listens for incoming peer connections. When a

connection is accepted, a list of all open data connections is updated. The final topology of the connections at the data exchange level is a mesh topology as shown in Figure 4-4.

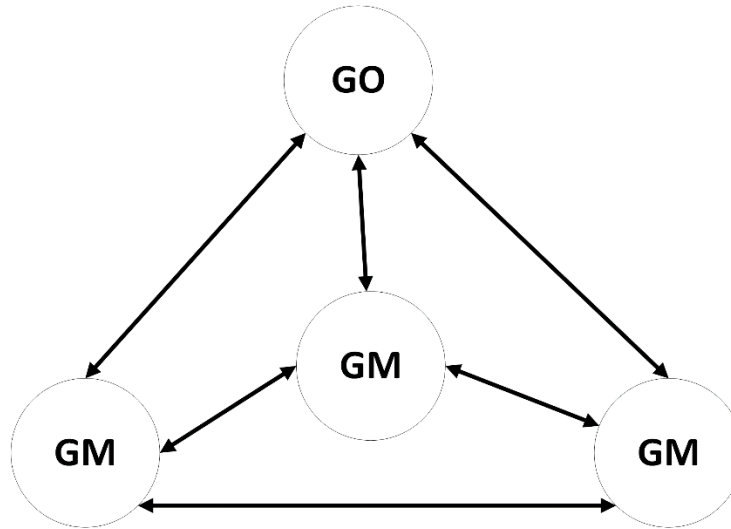


Figure 4-4 The topology for the data exchange connections.

4.1.2.2 GO and GMs Interactions

The group management protocol is a centralized one, where the GO is the entity that orchestrates it. The GO receives heartbeat messages from members, updates its peer list as necessary, and send the current list of peers to the other devices. These operations are carried out using the management connections that were opened previously.

4.1.2.2.1 Heartbeat Messages

The heartbeat messages are used to announce that the member is still alive. Each group member sends a heartbeat message every $T_{HeartBeatGM}$ seconds to the GO through the opened management sockets. An example of the heartbeat message is “2953112190.083090484,Nexus,aa:aa:aa:aa:aa:aa:aa:aa, 192.168.45.10”. It is a comma separated string composed of the concatenation of the uniqueID,

devicename/username, MAC address, and IP address. If the devicename/username contains a comma, it is padded by replacing the comma with the special characters $\langle\Diamond\rangle$ to avoid misinterpretation of the message. Upon receiving the message, the GO unpads the devicename/username if necessary and stores/updates the peer information in a special data structure.

4.1.2.2.2 *Announcing the List of Available Peers*

The group owner notifies the members in the group about the peer information by sending a message every $T_{HeartBeatGO}$ seconds that contains the list of all known peers, where $T_{HeartBeatGO}$ is a multiple of $T_{HeartBeatGM}$ to allow the GO to collect the data of more than one GM before sending the peers list message. The list is a semicolon-concatenated string that is composed of the heartbeat messages received from the GMs plus the information of the GO itself. Upon receiving the list from the GO, each GM stores/updates a peer list. It then attempts to open data-exchange connections with unconnected peer in the list (including the GO), using the IP addresses given in the list. A list of all open data-exchange connections is updated to reflect how many connections are open.

4.1.2.2.3 *Duplicate Connections Removal*

As the group members attempt to connect to each other as soon as they receive the list from the GO, there is a possibility that duplicate data-exchange connections are opened between them. To prevent this, we added a random wait time before a peer attempts to connect to another. Then, before connecting to a peer, a GM checks if a connection with such peer already exists or not. While this decreases the probability of

duplicate connections, it is still possible for duplicate connections to be established. To avoid this problem completely, we have added a new procedure that allows removing any redundant connections.

The duplicate connection removal procedure runs after a peer opens connections with other peers. Each peer iterates along the list of the data exchange connections, finds any duplicated connection (using the IP address associated with the connection), and removes it. To avoid removing the connection from both ends, we used the last octet value in the IP address of the device to guarantee only one connection removal. Each device when iterating through the list, it removes the duplicated connection only if the last octet in the IP address of device is higher than the last octet of the IP address of the peer associated with the connection. For example, if node A has the IP address 192.168.1.10 and node B has the IP address 192.168.1.20. The last octets for A and B are 10 and 20 respectively. When A tries to remove a duplicated connection with B , it finds that the last octet in these sockets is 20. Device A stops and does nothing in this case. In the same time, B attempts to remove the duplicated socket with A . Device B finds that its octet is greater than the octet associated with the sockets, so it closes one of the duplicated connections. After running this step, each device will keep only one data connection with other peers.

4.1.2.2.4 Pruning Peers

To tell whether a peer is alive or not, a time-to-live (TTL) value is associated with each peer in the stored peer list. The TTL value is initialized to P_{TTL} (where P_{TTL} is a multiple of $T_{HeartBeatGO}$ to allow the GMs to perform pruning and addition of peers at

the same step). This value is decreased periodically by all devices if the peer information is not heard again. If the GO receives a heartbeat message from the peer, it resets the TTL value for that peer to P_{TTL} . If a GM sees the peer again in the list transmitted by the GO, it resets the value to P_{TTL} again. Once the GO determines that a certain peer's TTL value has reached zero, it assumes that the peer has departed the group. The GO then disconnects from any data and management connection opened previously with that peer and remove it from the list of peers. In the next time the GO transmits a peer list message to its members, the removed peer will not be there. A GM continues to decrease the TTL value for the removed peer until it reaches zero. In that case the GM closes any data or management connections associated with that peer.

4.1.2.2.5 Restarting After GO Failure

If one of the GMs fails or drops from the group, the GO will tell other members that this peer is not available any more. However, in case of GO failure, the devices would not hear the normal peer list message. They will start to decrease the TTL for the GO. Once the TTL value for the GO reaches zero, they all disconnect from the GO. In this case, they detect the removal of GO and they flush any peer data structure and start over again.

A flowchart that shows the complete steps for the proposed group management protocol for both GO and GM is shown in Figure 4-5.

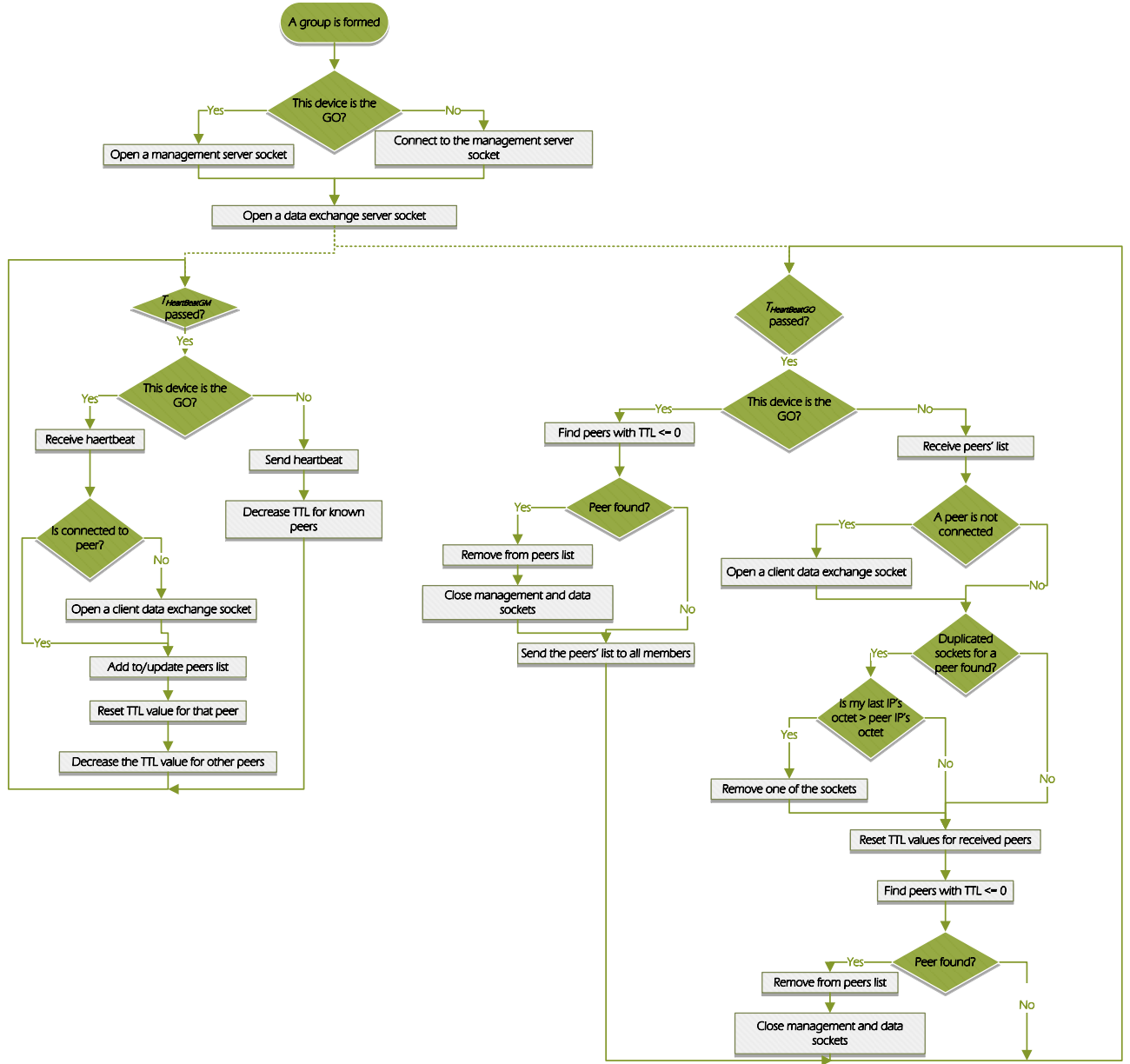


Figure 4-5 The flowchart for the group management protocol.

4.2 Implementation and Validation

We have developed two different Android P2P applications; during our implementation on Android we faced certain limitation that we opted to overcome as mentioned in the next subsection. The first application is a remote sensor streaming

application between two devices to demonstrate that Wi-Fi Direct could handle fast data streaming. To validate ELN, we have implemented a group chatting application, which utilizes the proposed protocol. Implementing ELN allowed only the devices that run this application to connect to each other, multiple peers to chat together, and the handling of addition or removal of peers seamlessly. Both applications are written in Java using the Android SDK.

4.2.1 Android Implementation Issues

The software support for Wi-Fi Direct in Android needs major expansion to enable P2P networking, thus we opt to make some progress toward that goal. Among the current shortcomings are the lack of accessibility to MAC and IP addresses for the device and the lack of a group management protocol that can handle topology changes and dynamic group membership. In addition, the members of a Wi-Fi Direct group know the IP address of the group owner, however they do not know the IP addresses of all other members in their group. Thus, a way of distributing the list of peers' IP addresses to every member is needed. Our proposed protocol overcomes such limitations. However, not all shortcomings are addressed in ELN. Specifically, the inability of a device to be associated with more than one group at the same time. The absence of this feature in Android limits the extension of the peer-to-peer system beyond the range of one group. Allowing multi-group communications is addressed by our proposed EMC protocol for multi-group data sharing, discussed in Chapter 5.

4.2.2 Remote Streaming of Sensors Readings

The goal of this application is to connect two Android devices and test how they could communicate over Wi-Fi Direct. This assures how Wi-Fi Direct is suitable for streaming data. The application collects the readings of the local accelerometer sensor and displays them in a local sensors graph. Once a group is formed, the two devices start to stream their local accelerometer readings to the other device. Each device listens for incoming sensor data and displays the readings in a graph.

The earlier version of this application was written in Eclipse then it is ported to the AndroidStudio. The application targets Android API level 14, which is the first implementation of Wi-Fi Direct in Android. Service discovery is not used as it is not supported at this API level. The choice of this API level is made to allow testing how fast and stable is the first Android implementation of Wi-Fi Direct. The application is made available at the Google Play Store [76]. A screenshot of the application, where one device is receiving and displaying the other device's readings, is shown in Figure 4-6.

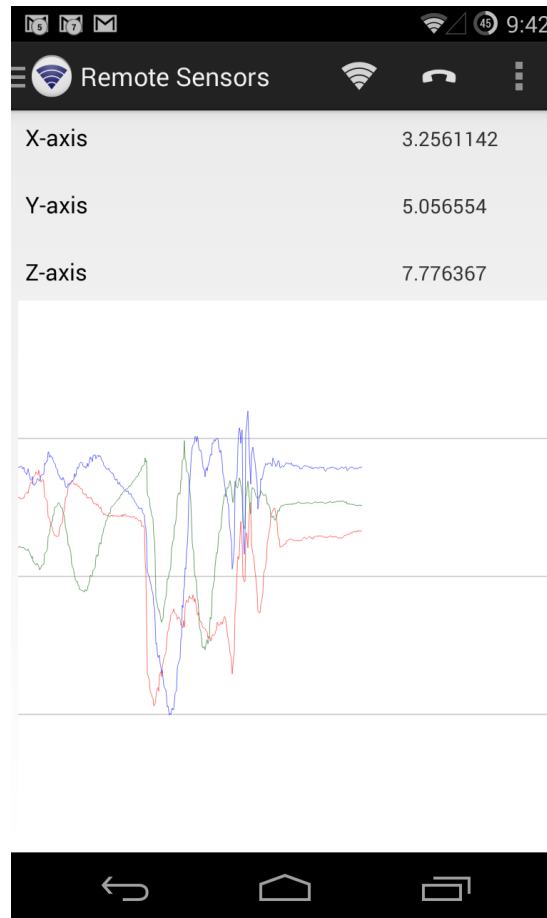


Figure 4-6 A device is receiving and displaying the sensor readings.

The application was tested in two Samsung Galaxy Tab 2 7.0 devices. The devices were able to connect together to exchange their accelerometer sensors' readings. Both of them were able to display the received data in the remote sensors graph. The streaming and displaying of the data was very fast and runs smoothly without any glitches.

4.2.3 Group Chatting

The goal of this application is to validate ELN. It allows multiple devices to connect to share chat messages. Any chat message sent from any device is sent to and displayed

on other devices in the group. If a new device is willing to participate in the chat session, it simply connects to the group and ELN handles the process of announcing its existence to current group members. The framework also opens the required sockets for the device to send and receive chat messages.

This application is written in AndroidStudio. It is published and available at the Google Play Store [77]. The first level of the Android API that support Wi-Fi Direct service discovery is API Level 16. Thus, the application targets the Android API level 16 in order to be able to run the service discovery mechanism used in the connection setup phase of ELN. The main module in the application starts the connection setup protocol and announces that it provides a chat service. It also searches for similar devices that provide the same service. As part of this protocol the application creates a service record that contains the uniqueID, the username (if the user had chosen one from the settings, otherwise the device name is used), and the user availability (for the sake of testing it always set to available). Once all data is collected, the service discovery announcement starts. At the same time the module listens for any incoming service announcements. Upon receiving a service announcement from a nearby device that provides the chat service, the module extracts the username and displays it in a list of discovered devices. The user can then pick one of the chat service providers and connect to it to. If there is no active Wi-Fi Direct group, the Android framework creates one.

The main module follows the same flow of events as described in the group management phase. It implements functions for finding the device own IP address,

compiling the device information in a csv string (i.e. the heartbeat message), producing a peer list message (for GO only), processing the incoming management, and displaying the incoming data (chat) messages. A helper NearByPeers class is used by the main activity to facilitate performing the group management protocol tasks. This class contains a list of all opened management sockets, a list of all opened data-exchange sockets, and a peer record. Several methods are provided in this class to add new peer record, connect to a peer, prune dead peers, and remove duplicate socket connections. The peer record itself is another class that composed of the attributes uniqueID, peerName, peerMacAddress, and peerIpAddress.

In addition, the main module uses SocketManager class to handle sockets. For every open socket, either a management or data socket, an object of the SocketManger class is created and associated with that socket. Whenever a message is received in a socket, the associated SocketManager object notifies the main module, indicates the type of socket, e.g., a management or a data socket, and provides a copy of the object to the main module. A ChatFragment is used by the main module to display received chat messages and to forward them to all nearby peers. Two timers are used to execute the periodic task defined in the management protocol like sending heartbeat, sending peer list, pruning disconnected peers, removing duplicate socket connections, and decreasing the TTL values for unheard from peers. Table 4-1 shows a brief description for what each component does.

Table 4-1 Application components and their description.

Component	Description
NearByPeers Class	<ul style="list-style-type: none"> - Attributes: Management Sockets List, Data Sockets List, and Peer Record - Methods: Add New Peer, Connect To Peer, Prune Peers, and Remove Duplicated Sockets
Main Activity	<ul style="list-style-type: none"> - Performs the service discovery announcement. - Finds nearby devices and adds them to a list. - Connects to a device, opens/connects to the required management sockets, and opens/connects to data exchange sockets. - Periodically sends peer info (GM) every second/ peers list (GO) every 5 seconds. - Periodically decreases the TTL values for peers every second. - Periodically prunes peers every 5 seconds. - Handles messages from management sockets and add/update peers list, prune peers, remove duplicated socket connections, and reset TTL values accordingly. - Opens a data exchange connection if not already connected
SocketManager Class	<ul style="list-style-type: none"> - Handles incoming data from the socket - Notifies the main activity about the data and its type (Management/ data) - Handles outgoing data to the socket
Chat Fragment	<ul style="list-style-type: none"> - Handles user text input and sends the chat message to all peers. - Displays chat messages from other peers.
Setting Activity	<ul style="list-style-type: none"> - Allows the user to change the announced username

The group chatting application is tested on 4 Android devices (two Samsung Galaxy Tab 2 7.0 tablets and two Nexus 4 phones). In this test, we fixed $T_{HeartBeatGM}$, $T_{HeartBeatGO}$, and P_{TTL} to 1, 5, and 30, respectively. We started the application in two devices first and connected them together. Each of the two devices was able to discover the other one, and to display its name in the list of discovered devices. The management sockets were created and connected. The heartbeat messages and the peer list messages flowed as expected. The GM was able to connect to the data sockets of the GO and vice versa, implying that the NearByPeers object was populated with the current peers. Sending

and receiving chat messages were going normally. We then started the application in the third device and performed the connection setup protocol. The third device was able to discover the GO of the current group and open socket connections (management and data). The device was successfully added to the list of the peers at the GO. The result was that every device was able to send a chat message to the other two. Finally, the forth device was connected successfully and the data exchange was running normally. Figure 4-7 shows a screenshot where three devices are chatting together. Finally, we disconnected one of the devices from the group to see how the other devices react and observed that they successfully removed the departing device from their NearByPeers object and closed opened sockets for that device.

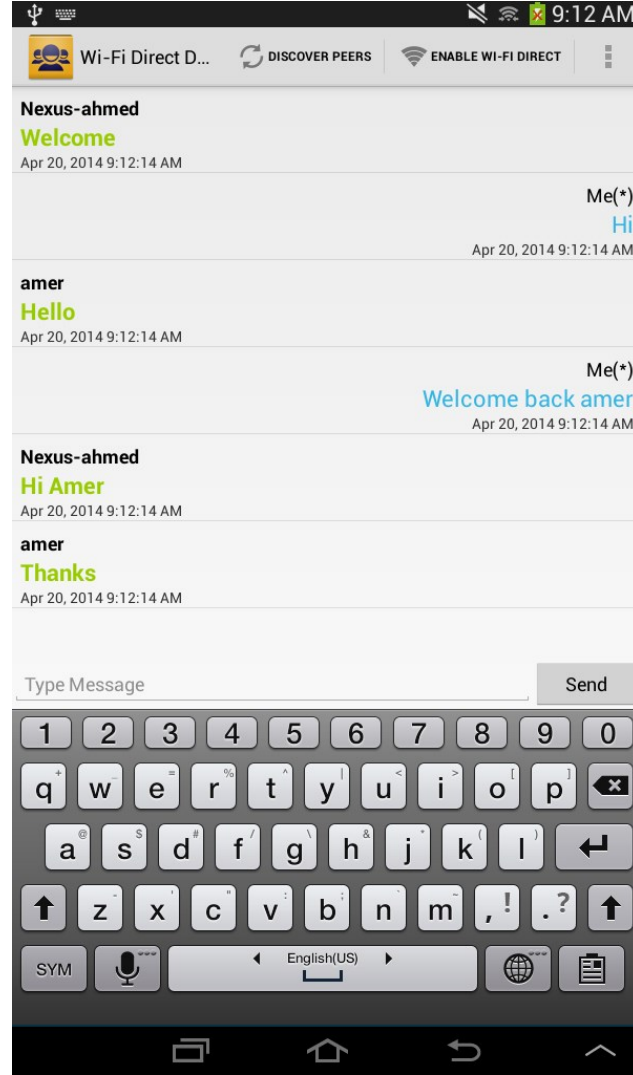


Figure 4-7 Three devices are chatting together.

4.3 Performance Evaluation

In this section, we evaluate the performance of the proposed ELN. Mainly, we focus on the group management phase. The goal of the evaluation is to capture the protocol overhead, and assess the ability of the protocol to adapt to topology changes. The assumptions used in this section are shown in Table 4-2. All the calculations performed next are based on the worst-case scenarios.

Table 4-2 The assumptions used in this section.

Parameter	Value
Maximum length of the heartbeat message	N bits
Maximum number of members in the group, including GO	M
Maximum length of peers' list message	N*M bits

4.3.1 Protocol Overhead

The GO sends a peer list message every $T_{HeartBeatGO}$ seconds and receives a heartbeat message every $T_{HeartBeatGM}$ seconds from every member. A GM sends a heartbeat message every $T_{HeartBeatGM}$ seconds and receives a peers' list message every $T_{HeartBeatGO}$ seconds.

$$\begin{aligned} & \text{Total number of message (sent and received) by the GO} \\ &= \frac{(M-1)}{T_{HeartBeatGM}} + \frac{1}{T_{HeartBeatGO}} \quad \text{messages per second} \end{aligned}$$

$$\begin{aligned} & \text{Total number of message (sent and received) at each GM} \\ &= \frac{1}{T_{HeartBeatGM}} + \frac{1}{T_{HeartBeatGO}} \quad \text{messages per second} \end{aligned}$$

The bandwidth consumed by the protocol is the number of bits transmitted and received per second by both GO and GM. Thus,

$$\begin{aligned} & \text{Bandwidth consumption by the GO} \\ &= \left(\frac{(M-1)}{T_{HeartBeatGM}} \times N \right) + \left(\frac{1}{T_{HeartBeatGO}} \times N \times M \right) \\ &= \left(\frac{M-1}{T_{HeartBeatGM}} + \frac{M}{T_{HeartBeatGO}} \right) \times N \text{ bps} \end{aligned}$$

Bandwidth consumption at each GM

$$\begin{aligned}
&= \left(\frac{1}{T_{HeartBeatGM}} \times N \right) + \left(\frac{1}{T_{HeartBeatGO}} \times N \times M \right) \\
&= \left(\frac{1}{T_{HeartBeatGM}} + \frac{M}{T_{HeartBeatGO}} \right) \times N \text{ bps}
\end{aligned}$$

If we assume that N , M , $T_{HeartBeatGM}$, and $T_{HeartBeatGO}$ equals 500, 20, 1, and 5, respectively, the total bandwidth consumption at the GO = 11.5 Kbps and the total bandwidth consumption at each GM = 2.5 Kbps. Assuming that the bandwidth for Wi-Fi Direct is 54 Mbps, we can conclude that the overhead of the protocol is negligible.

4.3.2 Topology changes

In this section, we assess how the protocol adapts to topology changes. In case of connecting a new member to the group, the GO starts hearing the heartbeat message from that member after $T_{HeartBeatGM}$ second. This means that after $T_{HeartBeatGM}$ second of connecting to the group, the GO and the new GM can start data exchange. Other members in the group knows about the addition of the new member once they receive, from the GO, the peer list message that comes after at most $T_{HeartBeatGO}$ seconds (additional $T_{HeartBeatGM}$ seconds should be taken into consideration, as the GO must wait $T_{HeartBeatGM}$ seconds before receiving the heartbeat from the new GM). Thus, the previous members and the new GM can start data exchange after $T_{HeartBeatGM} + T_{HeartBeatGO}$ seconds in the worst case.

Let us now consider the case of removing a member from the group. Assume that the period for decreasing the TTL value is one second. Both the GO and the GMs start decreasing the TTL value for the disconnected peer every second, as they are not hearing from it any more. The GO and GMs perform pruning of peers every $T_{HeartBeatGO}$

seconds and remove any peer with TTL less than zero. As long as the TTL value for the peer is greater than zero, the GO continues to include it in the peer list message. This causes the GMs to reset their TTL values for the disconnecting peer to P_{TTL} as they still hear about it. After P_{TTL} seconds, the GO finds that the peer is already gone, so it stops including it in the peer list. When the GMs receive the revised peer list message from the GO, they will have a TTL value of $P_{TTL} - T_{HeartBeatGO}$ for the disconnected peer (as they already started decreasing the value after receiving the last message for GO). It takes $P_{TTL} - T_{HeartBeatGO}$ seconds more before the TTL value reaches zero and the GMs remove the disconnected peer from the list of their nearby peers. Thus, after nearly $2P_{TTL} - T_{HeartBeatGO}$ seconds (P_{TTL} seconds for the GO to notice the disconnection and $P_{TTL} - T_{HeartBeatGO}$ more for the GMs), the removal of a peer will be reflected at all peers. It is worth noting that, handling the disconnection of many peers at once requires the same amount of time mentioned above for one peer.

The protocol also can handle the case when a peer is not able to communicate with others due to a temporary problem, like interference or jamming. In such a case the GO will not hear the heartbeat message, thus it starts decreasing the TTL value for the mentioned peer. The GMs also will Decrease the TTL value for that peer. If the peer can communicate again within $P_{TTL} - 1$ seconds, the GO will reset its TTL value. The GMs, in that case, will also reset the TTL value within $T_{HeartBeatGO}$ seconds of the GO re-initialization of the TTL value for the peer. Therefore, if the channel is jammed for a period less than P_{TTL} seconds, the group will be able to continue its operation.

If we assumed the values 1, 5, 30 for $T_{HeartBeatGM}$, $T_{HeartBeatGO}$, and P_{TTL} , respectively, adding new peer takes 1 and 6 seconds for the GO and GMs to handle, respectively. Removing a peer takes 30 and 55 seconds for the GO and GMs respectively.

4.4 Conclusions

In this chapter, we have presented, ELN, a new protocol for enabling peer-to-peer networking over Wi-Fi Direct in Android. The main components of the protocol are a connection establishment phase and a group management phase. The connection establishment phase enables only the devices with the same interest to connect. Meanwhile, the group management phase allows treating the Wi-Fi Direct topology, which is by convention a star network, as a mesh network. ELN does so by providing a mean of distributing the peer IP addresses, facilitating transport layer connections and managing addition and removal of peers from the group.

The proposed protocol can be applied to any type of applications including audio/video streaming, dissemination of traffic information, dissemination of emergency data, and last-mile connectivity. Two applications have been developed to validate the proposed protocol. The first is a remote sensor streaming application between two devices that validates the ability of Wi-Fi Direct in handling data streaming at high rate. The second is a group chatting application. By implementing ELN in this application, the devices that run this application could connect to each other, multiple peers could chat together, and the handling of addition or removal of peers was seamless. In chapter 7, we provide a unified simulator for all the protocols in this dissertation. By using this simulator, we were able to get insights about and

record the performance of ELN in many different scenarios, which is discussed in chapter 7.

Chapter 5: Efficient Multi-Group Formation and Communication Protocol for Wi-Fi Direct

In this chapter, we propose an Efficient Multi-group formation and Communication (EMC) protocol for Wi-Fi Direct [79][81]. EMC also dynamically and efficiently partition the devices into Wi-Fi Direct groups based on certain criteria, such as the battery specification. The concept of ADS is utilized to allow devices to share their information with nearby devices prior to creating the groups. A device with a higher rank than those in its range opts to create a Wi-Fi Direct group. Once a group is created, the group owner (GO) uses a service discovery record to distribute its credentials to nearby devices. A device that decides to be a group member (GM) should select one of the nearby GOs to connect to. Once a group is formed, the GO designates from its GMs what we refer to as proxy members (PMs) that link the group to other groups. Each PM uses its “WLAN” interface to join the group instructed by its GO. To avoid depleting the batteries of the GOs and to adapt for changes in the groups, a teardown signal is sent by a GO to notify the devices about restarting the EMC protocol. A typical topology for the network after running EMC is shown in Figure 5-1. An Android chat application is developed to validate EMC, where each device in a group can chat with other GMs in the group as well as other groups.

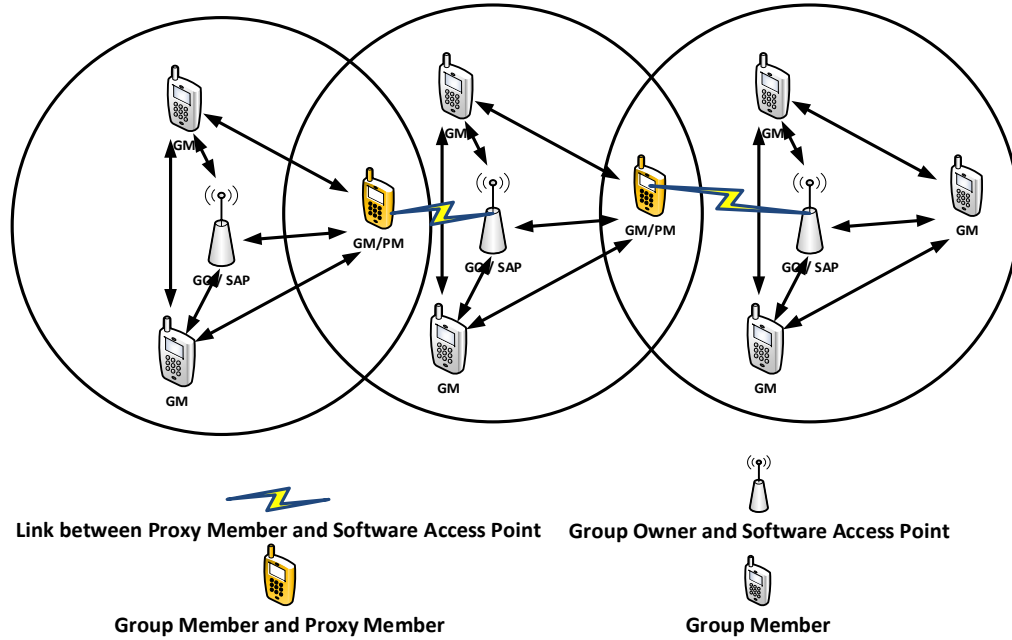


Figure 5-1 A typical topology for network after running EMC

5.1 Approach Overview

Enabling D2D communication in the situations where there is no available infrastructure is highly demanded in areas that have no cellular coverage, or suffered a massive power outage or the occurrence of natural disaster. EMC is geared for enabling the D2D connectivity in such application scenarios using Wi-Fi Direct. As the topologies may not be static in the mentioned scenarios, it is necessary to extend the protocol support of Wi-Fi Direct to allow dynamic group creation and multi-group communication. Basically, there is no mechanism for electing a GO among multiple devices based on certain criteria. In addition, broad dissemination of alerts needs to extend the spatial coverage of a group, which is not feasible due to the limited communication range. Increasing the spatial coverage requires the support of inter-

group data sharing. Moreover, devices may be interested in receiving alerts on multiple events and hence should be members of multiple groups. EMC overcomes such limitations through introducing the following features:

5.1.1 Support of Initial Data Exchange:

To enable effective GO selection, EMC defines certain information to be exchanged between nearby devices before interconnecting them. We utilize the same idea of using the service discovery in Wi-Fi Direct, mentioned before in ADS, to embed the information in local service records and to transmit them to nearby devices that query for available services. However, we do not exchange such information beyond one hop neighbors. There are two different local service records in EMC, one called DeviceInfo record that holds information regarding the device, and the other called SAP record, which is transmitted only by GOs. Figure 5-2 shows sample format of these two service records and the next subsections explain relevant entries in each record.

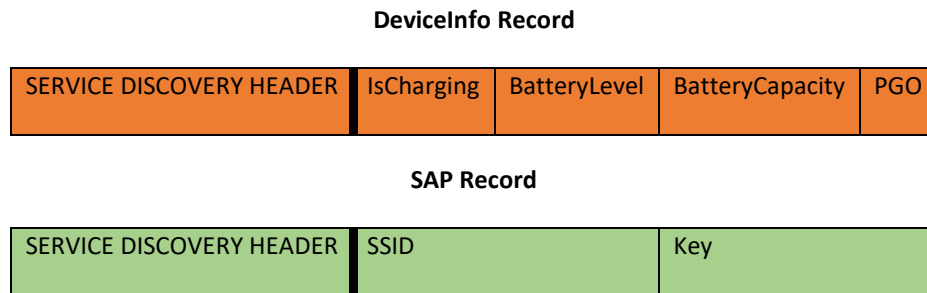


Figure 5-2 The format of the DeviceInfo and the SAP records

5.1.1.1 Battery Specifications

An example of the information exchanged by devices that run EMC as part of the DeviceInfo record may include 1) charging state, 2) battery level, 3) battery capacity. A charging device with a high battery level and a large capacity is a good choice for

being a GO. Thus, we could employ a ranking criterion based on such information as follows:

$$Rank = State \times \alpha + \frac{Level}{100} \times \beta + \frac{Capacity}{MaxCapacity} \times \gamma$$

Where the *State* is 0 or 1 depending on the charging state, *Level* has a range of [0, 100] reflecting percentage relative to the fully charged status, and the *Capacity* can vary based on the device. *MaxCapacity* should be chosen based on the specs of several commercially available devices. The weighting factors α , β , and γ reflect the importance of each term. Each device uses this mechanism to calculate the ranks of all devices in its range to assess the possibility of becoming a GO. GMs also use such ranking mechanism to choose the best group to join.

5.1.1.2 The Proposed GO

Nearby devices could get the rank of each other after exchanging their battery information. This rank is the main factor for deciding which device should serve as a GO. However, a device could falsely think that a neighboring device has a better rank than its own rank and concludes that such device should serve as a GO. Figure 5-3 explains such dilemma.

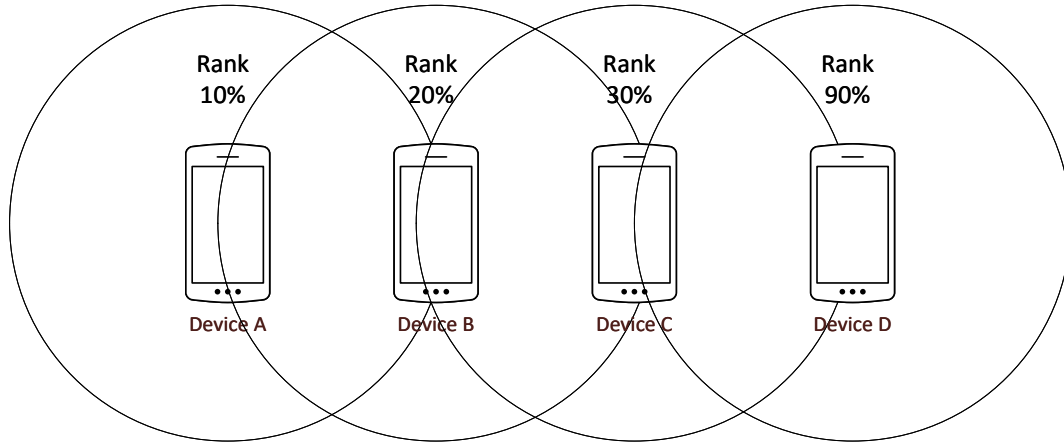


Figure 5-3 Devices Ranks as seen by reachable devices

As shown in the figure, *Device A* has a rank of 10% and *Device B* has a rank of 20%. Due to such ranks, *Device A* finds that *B* has a better rank, thus concludes that *Device B* should be a GO and *A* should be a GM in the group of *B*. At the same time, *Device B* itself sees that *Device C* should be a GO due to its better rank. Thus, *Device B* concludes that it should be a GM in *Device C*'s group. The same applies to *Device C* and *D*. Finally, the only device that becomes a GO is *D* and devices *A*, *B*, and *C* should be GMs now. For *Device C*, the assignment is fine, as it has a GO, *D*, on its range. Thus, *Device C* could now join the group of *Device D*. However, this is not the case for *Device A* and *B*, as the groups they thought that they could join do not exist. Moreover, the two devices, *A* and *B*, cannot reach *Device D*. Thus, they will become orphaned members.

To overcome the mentioned problem, we add a field called the Proposed Group Owner (PGO) to the service discovery record (DeviceInfo record) that the devices exchange. The PGO field is used to denote the device amongst the reachable neighbors

that is suggested to serve as a GO. So, a device when exchanging its battery information with others it also tells them which device it suggests as a GO using the PGO field. This allows the devices to know if there are other devices that have better rank that the sender device sees that they cannot see. Returning to our previous example, *Device B* tells *Device A* that its PGO is *Device C*. Now *Device A* knows that *Device B* has a better ranked neighbor so it will not be a GO. Thus, *Device A* decides to become a GO (i.e., its PGO now is *A* itself). Similarly, *Device C* tells *Device B* that its PGO is *Device D*, so *Device B* knows that *C* cannot assume a GO role. Thus, *Device B* changes its PGO to *B*. After a second round of service records exchange, *Device B* tells *Device A* that its PGO is *B*. *Device A* finds that *B* has a better rank so it declares *B* as its PGO also. The result is that we will have two groups and two group members, thus we avoided having orphaned members.

5.1.1.3 SAP Credentials

To connect to an access point, a device usually needs the SSID and the Key. As the Software Access Point (SAP) created by a GO can be treated as a normal access point, knowing the SSID and the Key for such SAP is sufficient for a device to connect to it. EMC embeds the SAP credentials (SSID, and Key) into a service discovery record that we call SAP record which is shown in Figure 5-2. The SAP record is used to inform devices about nearby groups, and to allow a proxy member to connect as a legacy client to the group chosen by its GO. As the service records are sent in clear text, malicious devices can steal the SAP credentials. To protect a Wi-Fi Direct group from malicious devices that can associate with it as legacy clients, EMC encrypts that SAP Key before

embedding it in a service record. EMC-enabled devices only can decrypt the SAP key in the service record.

5.1.2 Support of Intra and Inter Group Communication

Once a group is created and populated with members, there is a need to enable intra-group communications. To do so, we utilize our ELN protocol, described in Chapter 3, to manage the devices in a group. Basically, the GO opens a management connection with each new member. The list of all devices connected to the GO is distributed to the GMs to allow them to open data connections with each other.

We enable multi-group communication by associating qualified proxy members (PMs) to their group using the “p2p” interface and to another group as legacy clients using their “WLAN” interface. Unlike the work of Duan et al. in [53], which assigns the role of PMs to the GOs themselves, EMC assigns such a role to the GMs. The GMs, thus, serve as PMs between their original groups and the groups they are connected to as legacy members. Such policy employed by EMC allows a group to cover a larger area without imposing a constraint on where the GO can be, something that constitutes a major limitation in [53]. Another Feature is that EMC creates the groups dynamically which means that the legacy client selections are not fixed.

5.1.3 Insuring Network Connectivity

The number of available members that can work as PMs is limited, as not all the GMs are in the overlapping region between groups. Thus, the selection of PMs that connect multiple groups could have a wide impact on the final connectivity of the

whole network. Consider the network shown in Figure 5-4, there are three groups of A , F , and J . Each of these groups has certain GMs that are in the overlapping regions.

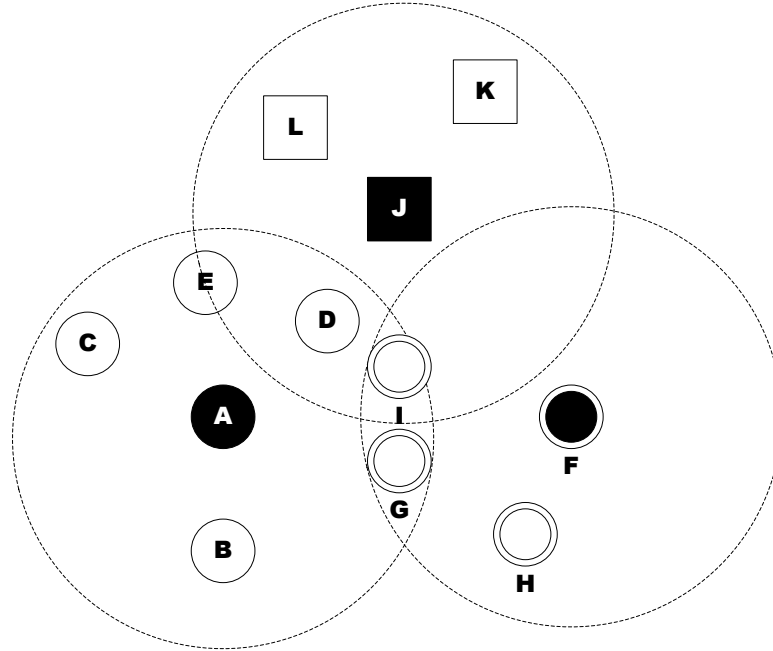


Figure 5-4 Three Wi-Fi Direct groups need to have PMs to be able to connect to each other. Devices with dark shade denote GOs. Each group has its shape symbol.

Let us consider the group of F , there are two GMs, I and G , that can connect this group to the group of A and the group of J . We note that I can reach both A and J , whereas G can only reach A . If F selects I as a PM for reaching the group of J and selects G to cover the group of A , the result will be a connected network (i.e., have one connected components). However, if F selects I as a PM for the group of A , the group of J will not have any PM to cover it. Thus, the network will not have a complete connectivity.


Due to the importance of PMs assignment, we have employed a combinatorial optimization algorithm called the Hungarian (Munkres) Algorithm [82] to solve the assignment problem in polynomial time. The Hungarian method is originally used for assigning agents to tasks by compiling a cost matrix with the columns denoting tasks and the rows denoting agents. An example of a cost matrix is shown in Figure 5-5. The algorithm then selects for each task an agent while minimizing the total cost. There is another variant of the algorithm that maximizes the assignment cost also. For our work, we refer to the tasks as groups and the agents as GMs. The cost matrix is produced using the ranks of the devices, that we mentioned before. We are interested in maximizing the total cost of assignment, as we favor devices with the highest ranks.

	Task1	Task2	Task3
Agent1	10	20	30
Agent2	5	10	15
Agent3	15	5	20

Figure 5-5 An example of a cost matrix where the minimum cost assignments for each task are colored green.

To compile the cost matrix, a GO first enumerates a list of the surrounding groups that at least one of its GMs can reach. For each group, the GO enumerates a list of the GMs that can reach it. Using those lists the GO can prepare the cost matrix. If there is a GM that can cover more than one group, the cost entries for these groups will have the same cost for the given GM, as shown in Figure 5-6. However, the generated cost matrix may not be complete, since not every group is reachable from every GMs. We substitute for the missing entries with a very small number to indicate that we do not

need these entries to be selected, as shown in Figure 5-6. If one of these entries is selected after executing the algorithm, we discard them.



	GO1	GO2	GO3
GM A	0.4		0.4
GM B		0.5	
GM C	0.3	0.3	

	GO1	GO2	GO3
GM A	0.4	-9999999	0.4
GM B	-9999999	0.5	-9999999
GM C	0.3	0.3	-9999999

Figure 5-6 An example of a cost matrix that a certain GO could have. On the left, we see that certain GMs can cover more than one group, thus we see in their rows the same cost repeated. On the right, we see the missing entries in the matrix been fixed by adding a very small value.

Given the cost matrix, we apply the Hungarian algorithm to get the assignment that maximizes the total cost. Such assignment should give the best connectivity between groups given that there is enough number of GMs that can work as PMs.

5.2 EMC Protocol

The EMC protocol is divided into multiple stages. The first stage is to choose GOs among the candidate devices. The second stage is for the GOs to create groups and distribute the credentials for SAPs. Next, the remaining devices choose which groups they should join. Then, the GOs designate from their group members PMs that link the members to other groups. Finally, to balance the energy consumption GOs send teardown message to inform all devices about restarting the protocol. This stage enables load balancing and allows the rotation of the GO role among energy-rich devices. A state diagram that shows the various stages that a device passes through is shown in Figure 5-7. The devices start with the deciding state and return to this state after the teardown process.

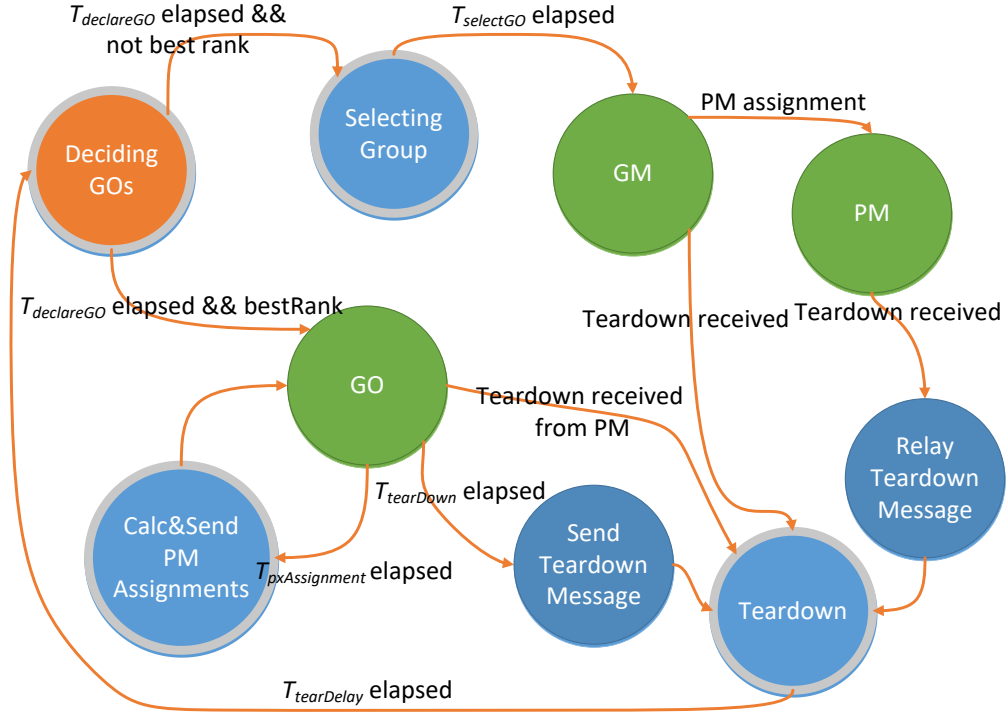


Figure 5-7 A state diagram of EMC

5.2.1 Choosing Proposed GOs

Each device starts by creating a local service record that contains its battery information mentioned and the Proposed GO (PGO) that are mentioned in section 5.1.1. The PGO is initialized to null at the beginning to indicate that there are no GOs that we heard from yet.

During this stage, a device D_i sends service discovery requests to reachable devices every $T_{sendInterval}$, as mentioned previously in ADS. Upon receiving D_i 's requests, nearby devices respond by sending back the stored record that contains their battery information and the PGO. The responses are collected and stored in a data structure that contains the IDs, battery information, and the PGOs for neighboring devices. D_i

then uses the battery information that it has received to calculate the ranks of the nearby devices, as discussed in section 5.1.1.1, and hence updates its PGO. If no device around has a better rank higher than itself, then PGO is updated to reflect that D_i itself is the PGO. We note that when D_i compares the ranks of the nearby devices it only considers devices that have their PGO pointing to themselves. By doing this we eliminate the possibility of becoming orphaned as discussed in 5.1.1.2. D_i stay in this stage for a period $T_{declareGO}$. Selecting $T_{declareGO}$ is a trade-off between collecting sufficient information from neighbors and quickly creating groups to allow data exchange. It is envisioned to select $T_{declareGO}$ based on the device density. A pseudo code for this step is shown in Figure 5-8.

```

phase1: //Choosing Proposed GO
foreach(device in network)
    createLocalDeviceRecord();
    requestDevicesInfo();
    while (elapsedTime <  $T_{declareGO}$ )
        if (DeviceRecord is received)
            store(DeviceRecord);
            calculatePGO();
        if (SAPrecord is received)
            store(SAPrecord);
            goto phase3;
    goto phase2;
end

```

Figure 5-8 Pseudocode for selecting candidate GO step.

5.2.2 Creating Groups

After the $T_{declareGO}$ period, the devices know who the PGOs are. A device that has PGO pointing to itself is then autonomously creates a group and becomes its owner. Before moving to the next stage, each GO adds a new local service record that contains the credential, i.e., SSID and Key of the SAP, which a legacy client can use to associate

with the group; for simplicity, we call it SAP record. If the device finds that it has a PGO other than itself, it declares itself as GM and proceeds to the next stage. Pseudo code for this step is shown in Figure 5-9.

```

phase2: //Creating groups
foreach(device in network)
  if(PGO == device.id)
    meIsGo = true;
  if(meIsGo)
    createAutonomousGroup();
    createSAPrecord();
    goto phase4;
  else
    declareGM();
    goto phase3;
end

```

Figure 5-9 Pseudocode for creating groups step.

5.2.3 Selecting a Group to Join

This stage lasts for a period of at least $T_{selectGO}$ where each device that is not a GO starts requesting service records from its reachable devices. Receiving a SAP record from a device D_i means that D_i is a GO. The collected records are used to update the previously populated structure to indicate for each known device its SAP record, if exists. Once the $T_{selectGO}$ period is elapsed, each non-GO device selects a GO, i.e., a device that has a SAP record, to connect to its group. Note that the SAP records are not used to connect to the groups as legacy clients, but they are stored to be used in the next stage of selecting PMs.

It is important to note that the devices are not required to run EMC at the same time. A device that starts the protocol and finds that a SAP record is available goes directly to the current stage to select a group. That is to avoid wasting time in the stage of

deciding to become a GO or not. If the device is better qualified for the GO role, EMC accommodates for that by restarting the protocol every certain period. Pseudo code for this step is shown in Figure 5-10.

```

phase3: //Selecting groups
foreach(declaredGM)
    requestSAPinfos();
    while(elapsedTime <  $T_{selectGO}$ ) begin
        if(SAPrecord is received) begin
            store(SAPrecord);
        end
        bestRank = -1;
        foreach(SAPrecord)
            rank = calculateRank(device in SAPrecord)
            if(rank > bestRank)
                bestRank = rank;
        end
        connectToGOof(bestRank);
        openSocketConnection();
        sendToGO(stored SAPrecords);
    end
end

```

Figure 5-10 Psedue code for selecting groups step.

5.2.4 Selecting Proxy Members

After a GM joins a group, it prepares a list of all the groups (SAP records) in its vicinity in order to be sent to its GO. Each GO after creating a group waits for $T_{selectGO}$ to allow possible members to join then it stays for a period of $T_{pxAssignment}$ listening for the lists of the reachable groups from its GMs. As mentioned in ELN, a GM send a heartbeat message every $T_{HeartBeatGM}$. In EMC, the GMs embed the list of reachable groups in the heartbeat messages of ELN. The received lists are then processed by the GO and stored in a special data structure that contains for each GM what are the groups (SAPs) it can reach along with the credentials for accessing these SAPs. In ELN, a heartbeat message is sent from the GO to the GMs each $T_{HeartBeatGO}$. EMC utilizes that time to calculate for each SAP a GM that act as a proxy member to that SAP's group.

Then the GOs, in EMC, embed the assignment for chosen members in their heartbeat messages. The selection of proxy members is based on the Hungarian (Munkres) method as discussed in section 5.1.3. Once $T_{pxAssignment}$ elapses, each chosen GM reacts by connecting to the indicated SAP as a legacy client using the credentials it already collected in the previous step; this GM is now denoted as PM. Pseudo code for this step is shown in Figure 5-11.

```

phase4: //Selecting Proxy Members
foreach(GO in network)
    while(elapsedTime <  $T_{pxAssignment}$ )
        receiveListOfGroups();
        processReceivedLists();
        selectPMs();
        sendPMassignmentsToMembers();
    end
end
foreach(selectPM)begin
    connectToChoosenSAP(SSID, key);
end

```

Figure 5-11 Pseudocode for selecting proxy members step.

5.2.5 Teardown and restart

A GO is involved in every intra-group interaction and may thus deplete its energy at a faster rate than GMs. To balance the energy load on GOs, EMC instruments periodic teardown of groups so that groups are reformed using fresh network state. A GO waits for a period $T_{tearDown}$ before starting to tear down its group by sending teardown messages to its GMs. The selection of $T_{tearDown}$ is subject to trade-off between having a balanced load on the devices and frequently incurring the group formation. EMC compensates for timing difference and allows devices to synchronize and restart the protocol at the same time. This is done by making each PM relay the teardown message

to the GO of other groups that this PM is connected to it as a legacy client. If a GO receives the teardown message from the PM before its $T_{tearDown}$ period ends, it may decide to follow through and inform its member about the teardown. Each device receives the teardown message waits a time $T_{tearDelay}$ to ensure that other groups are informed before processing the teardown. Afterwards, the devices restart the protocol. It is worth noting that the sequential teardown of groups will make sense when they do related tasks. The collective teardown is a means to synchronize them in their next run for EMC to form better groups, i.e., based on an updated node status. The pseudo code for this step is shown in Figure 5-12.

```

phase5: //Teardown and restart
foreach(GO in network)
    while(elapsedTime <  $T_{tearDown}$ )
        if(PM forwarded teardownMessage)
            break;
        sendTeardownMessageToGMS();
        while(elapsedTime <  $T_{tearDelay}$ );
        teardown();
        goto phase1;
    end
foreach(GM in network)
    if(GO sent teardownMessage)
        teardown();
        goto phase1;
    end
foreach(PM in network)
    if(GO sent teardownMessage)
        forward teardownMessageToSAP();
        while(elapsedTime <  $T_{tearDelay}$ );
        teardown();
        goto phase1

```

Figure 5-12 Pseudocode for teardown step.

5.3 *EMC Implementation*

5.3.1 Wi-Fi Direct Multi-Group Chat Application for Android

An Android chat application is developed in AndroidStudio to validate the EMC protocol; a version of this application is available at GitHub². The application used the API level 16 to allow the devices to use the service discovery in Wi-Fi Direct. The application is not required to run on each device involved in the protocol at the same time, but once it runs a device should be able to chat with devices in its group as well as devices in other groups without any manual interaction. A device would be either a GO, a GM, or a PM. A GO maintains its group using management sockets. Each device in the group exchanges chat messages using designated data sockets. PMs use special proxy sockets to exchange management and chat messages with other groups. Figure 5-13 shows screenshots of the app while it is running on two different devices. Although the app works fully autonomously, we have added manual override buttons on the top of the app to allow us to test specific parts.

² <https://github.com/ashahin1/EfficientWiFiDirectMultiGroups>

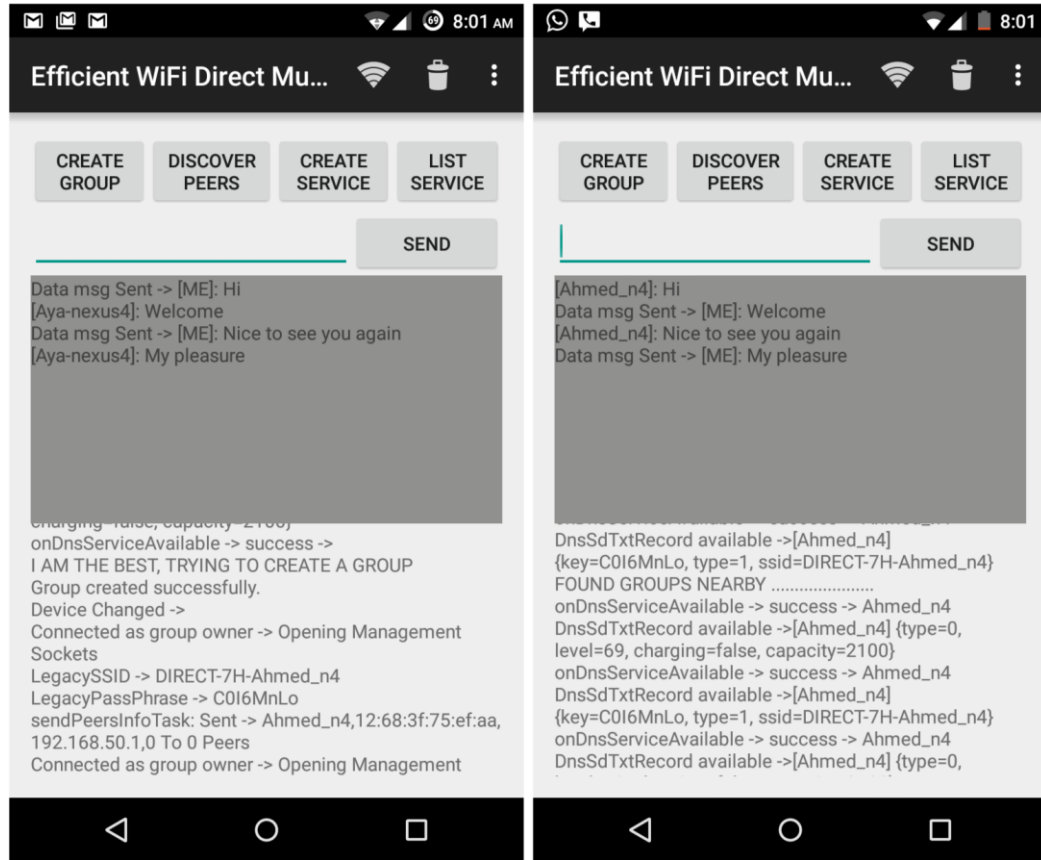


Figure 5-13 Screen capture of two devices running EMC.

5.3.2 Android Framework Modifications

Due to the support limitation for Wi-Fi Direct in Android, we were not able to have the app work out of the box. As all groups share the same 192.168.49.x subnet and all GOs share the same 192.168.49.1 IP address, there is no way to have PMs open bidirectional socket connection with devices in their group and other groups. In [53] this issue has been overcome by having certain members do broadcasts. We did not find such an approach an appealing solution though and chose to modify the source code for Android to allow the devices to have different subnets. For the sake of testing

the functionality of EMC, a static subnet assignment patch is applied to the Android versions running on the devices involved in the validation. In the next Chapter, we are proposing a subnet negotiation protocol, ISNP, that makes the assignment of subnet dynamic and allows EMC to scale well with large number of groups. Another issue in Android is that any request to join a group should be confirmed by the device that have a GO device. To allow fully autonomous operation, we changed the Android source code to allow the GO device to accept connection requests automatically.

5.3.3 Test Cases

Tests are performed using five devices, two Nexus 4, two Samsung Galaxy Tab 2, and one Asus Transformer tf700t. One of the Nexus 4 devices is kept with its original ROM (Android v5.1) with no modifications. The other Nexus 4 device is reloaded with CyanogenMod v12.1, (which is based on Android source code) after modifying it. The Samsung devices are running the stock ROM (Android v4.2.2). Since we do not have a complete source code for that specific Samsung model, we have disassembled the Android framework in those devices, applied the modification, reassembled, and then placed it back on the devices. The Asus device is kept running its stock ROM (Android v4.2.1). All devices are capable of running Wi-Fi Direct along with the service discovery. All tests were conducted in our research lab, where many other Wi-Fi access points exist. During our tests, we noticed that the interference was so high that some service discovery records were dropped. Using a Wi-Fi Analyzer app, we found that all the Wi-Fi channels are overwhelmed as shown in Figure 5-14. However, EMC was still able to run.

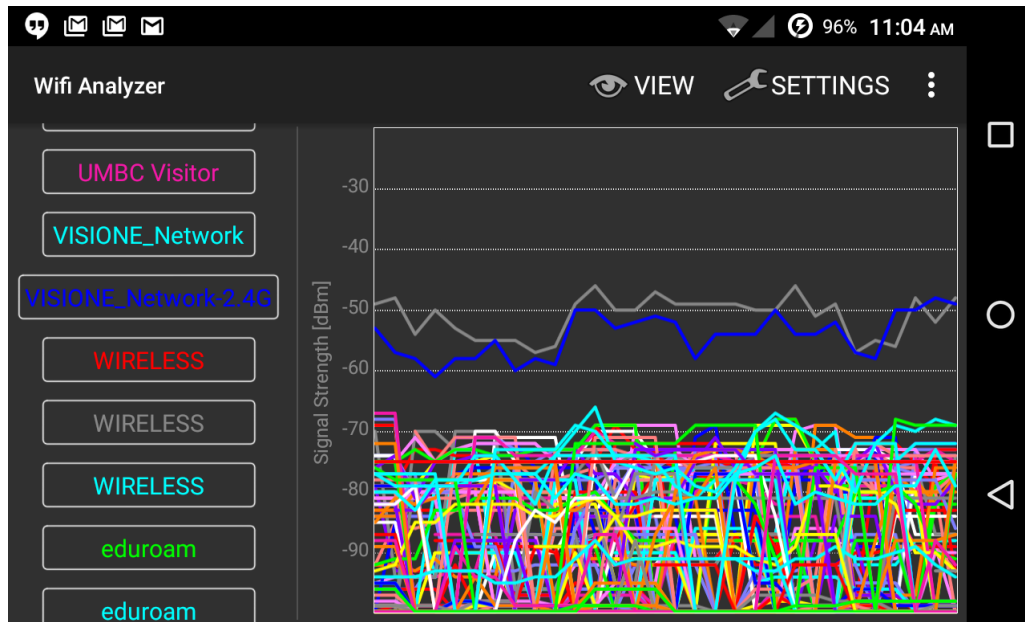


Figure 5-14 Analysis of wireless interference observed during the test..

5.3.3.1 Test Case 1: Group creation

In this test, we first used three devices (one Nexus 4 and Two Samsung Tab). Android reported 2100 and 1750 for their battery capacity, respectively, which is strange as the tablet should have more battery capacity than the phone (it should be a bug in Android). We kept the devices at nearly 100% battery level. The Nexus 4 device was intentionally plugged in an AC source. The chat app was then run in the three devices and we monitored EMC steps on the three devices. The Nexus 4 found that its rank is better than the other two devices, so it created a group. The two Samsung devices discovered that another device ranked better so they waited to select a group. As there was only one group created, the two Samsung devices chose it, which is the group created by the Nexus 4, and connected to it. The Nexus 4 did not choose any of the Samsung devices to serve as a PM as there is no other groups. At the end, the three

devices were able to chat together. After the predefined time for teardown, the Nexus 4 device asked its members to teardown. All of them then restarted the EMC protocol again. The same results were obtained when we ran the app in the five devices, where the Nexus 4 was the only one that was charging.

We then tested the incremental execution of EMC in the five devices by allowing a device to run the app then after some time another device runs it and so on. We started by one of the Samsung tablets, which declared itself a GO and created a group, as there were no other devices nearby. The remaining devices ran the app consecutively. The first device found that at least one group had been already established (as it received some SAP records), hence it started to select the best group. As there is only one group, the device selected and joined it. The same were done by the other devices. At the end, all five devices were able to chat together. We note that there are no PMs assignments, as no other groups were existed. The teardown mechanism performed well in this case and all of them restarted EMC again.

5.3.3.2 Test Case 2: Multi-Group Communication

In the first test, we validated that the EMC protocol is able effectively handle group creation. For the multi-group communication test, some devices should be outside of the range of others to have them form other groups. Therefore, we needed more devices and an open space to validate the whole steps of creating separate groups, selecting the groups, and selecting the PMs. To overcome the device count limitation and space constraints, we forced two devices to create two distinct groups and then tested how the other devices choose which of these groups to join. Thus, we added some extra

control in the app to allow a device to bypass the first step in EMC and proceeds to creating a group. We then used a Nexus 4 and a Samsung Tab that have been modified so that they can have different subnets for their Wi-Fi Direct groups, and got them to create groups. The Nexus 4 was kept charging, and thus it was the best GO, consequently the other devices selected the group of the Nexus 4 to connect to. The final distribution of devices was a Nexus 4 as GO with three other GMs and a Samsung Tab as a GO with no members. As all the GM devices with the Nexus 4 have heard from the Samsung Tab its SAP record, they knew that another group existed. After connecting the Nexus 4 group, the three devices notified the Nexus 4 about the group of the Samsung Tab. Once the proxy member selection period has elapsed, the Nexus 4 selected the GM Samsung Tab to serve as a PM. The GM Samsung Tab connected to the SAP of the GO Samsung Tab and became a legacy device in its group. As a result, the devices in the Nexus 4 group were able to chat with the GO Samsung Tab. After finishing the teardown period, the Nexus 4 sent a teardown message that reached the GO Samsung Tab. We noted that all devices responded by restarting the EMC protocol.

5.4 Performance Analysis

In this section, we analyze the performance of EMC during group formation and multi-group communication. For group formation, we are assessing how fast a group could be formed. The multi-group formation case we analyze the time needed for two groups to get connected and start data transfer. Such analysis gives an insight of the suitability of EMC for data sharing scenarios that needs fast connection times. Of

course, with careful selection of EMC parameters it can suite wide range of applications.

5.4.1 Group Formation

Let us assume that all devices start EMC at the same time. A device with a high rank declares itself as a GO and creates a group in a time $T_{declareGO}$. Other devices in its neighborhood enters the group selection mode. Being in this state a device waits a time $T_{selectGO}$ to collect SAP records. $T_{selectGO}$ should be large enough to allow the reception of SAP records from all the neighboring groups. The time needed before joining a group is only $T_{selectGO}$. This means that the time T_c a certain device needs to start data exchange with others is

$$T_c = T_{declareGO} + T_{selectGO}$$

Waiting for $T_{declareGO}$ is not needed for a device that runs EMC after having groups already created. Generally, $T_{declareGO}$ should not be too high to allow fast data exchange; yet at same time it should be long enough to account for possible loss of service discovery packets due to interference. The selection of $T_{selectGO}$ follows the same rules as in $T_{declareGO}$. EMC allows devices to join EMC at any time; thus, a device with better rank in a certain area may come after group creation in such an area. The device in this case will choose a group to join. The teardown signal that comes after $T_{tearDown}$ allows better adaptation to such a situation. Thus, $T_{tearDown}$ should be chosen wisely to allow better adaptation and in the same time less frequent restarts.

5.4.2 Multi-Group Communication

The GOs waits for a time $T_{pxAssignment}$ before sending PM assignments. Assuming that EMC is executed at the same time by all devices, a group would be able to exchange data with other groups after a time $T_{pxAssignment}$ from its creation. Thus, the time T_p needed for achieving inter-group communication is $T_p = T_{declareGO} + T_{pxAssignment}$. The selection of $T_{pxAssignment}$ is not trivial though, as it should be large enough to allow a GO to collect enough information from its members before deciding the best PM assignments. It is worth noting that if a device runs EMC after having the best GO in its range sent its PM assignment, it will never be assigned the PM role even if it is a better candidate for connecting to another group. The teardown process that comes after $T_{tearDown}$ accommodates such situation. The selection of $T_{tearDown}$ though is a trade-off between frequent restarting of EMC and better topology adaptation.

5.5 Conclusion

In this chapter, we have presented EMC, a protocol for creating Wi-Fi Direct groups dynamically by electing group owners based on the device energy reserve. EMC also enables connecting the created groups by selecting PMs to relay the data from one group to another, thus allowing multi-group communications. A chosen PM connects to another group using its “WLAN” interface, where it connects as a legacy client to the SAP of that group. EMC uses the service discovery protocol in Wi-Fi Direct to: (1) perform the distribution of the battery information that is used to rank devices along with the Proposed GO field, where devices with the highest rank in a certain area opts to create a group, and (2) distribute the SAP credentials of groups to their neighbors,

which allows PMs to connect to such SAP. To avoid depleting the battery of the GOs, EMC restarts itself after a certain period to allow rotation of the GO role.

An Android application is created to implement the EMC protocol. Certain modifications are done to the source code of Android to allow the groups to be in different subnets and to allow the automatic acceptance of connection requests. The applicability of EMC has been validated through testing the created application on five smart devices. The operation of EMC is analyzed to get an overview of its performance. The analysis has provided guidelines on how to choose the EMC parameters to achieve low-latency group creation, and battery optimization. To fine tune the parameter settings for EMC, we have performed several simulation experiments that are reported in chapter 7. We have also tested the performance of EMC against other approaches in that chapter.

Chapter 6: IP Subnet Negotiation in Wi-Fi Direct for Seamless Multi-Group Communications.

In this chapter, we present IP Subnet Negotiation Protocol for Seamless Multi-Group Communications (ISNP), which overcomes the limitation of Android that lets all the Wi-Fi Direct created groups to share the same range of IP addresses. As shown in Figure 6-1, such limitation causes IP address collision between devices in different groups. ISNP integrates with EMC by taking advantage of first phase of EMC to allow participating devices to negotiate distinct IP subnets with other devices before forming the groups. Once the groups are created by EMC, each GO uses its proposed IP subnet to assign IP addresses to the devices in its group.

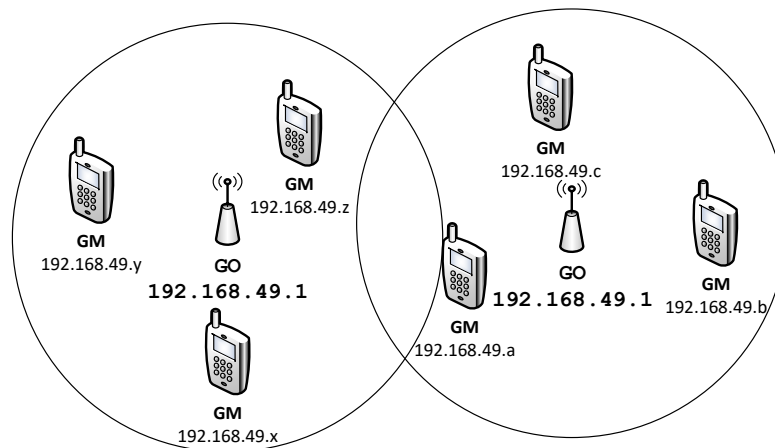


Figure 6-1 Two adjacent Wi-Fi Direct groups sharing the same IP subnet.

ISNP has two components, one that runs at the application level and utilized the service discovery records of EMC to negotiate the proposed subnets and another component that runs at the operating system (OS) level. The OS level component is

required to force Android to replace the default fixed IP subnet with the device's selected one. To accommodate for the devices with locked versions of Android that cannot be modified, ISNP allows devices with no modifications in their OS to still participate using their application level component. Basically, a device will still be able to propose and negotiate IP subnets with nearby devices using ISNP's application level module; however, such a device should be excluded from serving as GO by EMC. The application level module of ISNP is validated by integrating it with our previous Android application that we wrote for EMC. For the OS level module, we have modified the source code for the devices involved in testing and then compiled and uploaded the new version of Android on these devices.

6.1 Problem Statement

When connecting multiple Wi-Fi Direct groups using relay nodes, the Wi-Fi Direct implementation on Android will not make it possible for these groups to share data in a bidirectional way. The reason for this is that the fixed assignment of GO's IP address and the DHCP address range cause all the groups to fall into the same IP subnet. Thus, the devices in these groups will have collisions in their IP addresses, which will hinder them from having bidirectional connections at the transport layer. Overcoming such limitation provides a possibility for realizing full multi-group communications in Wi-Fi Direct. Obviously, it is not practical to assign different subnets for devices manually. To have a seamless group formation that can scale, a dynamic assignment of the IP subnets for different groups is warranted. ISNP opts to fill such technical gap. Upon

integrating ISNP with EMC, the IP subnets of any two overlapping groups would be different as shown in Figure 6-2.

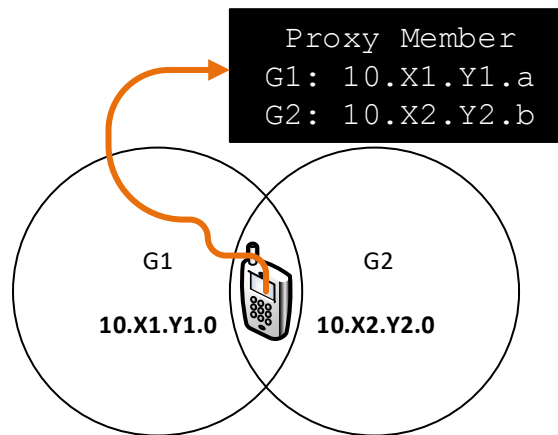


Figure 6-2 IP subnets for two adjacent groups after integrating ISNP with EMC

6.2 The ISNP Protocol

ISNP enables adjacent groups to have different IP subnets, which makes it possible to have bidirectional communications between groups at the transport layer. ISNP leverages the service discovery records of EMC to provide a connectionless negotiation of the IP subnets; a choice that makes it very lightweight and efficient. In addition, the integration between ISNP and EMC provides a complete data sharing solution using Wi-Fi Direct.

6.2.1 ISNP Overview

ISNP is composed of two modules, the first runs at the application level and the second is handled at the OS level. The application-level module utilizes the service discovery records of EMC to allow the devices to propose IP subnets and announce them to nearby devices. It also makes sure that conflicts in the proposed subnets are

resolved correctly. The second part assigns the proposed IP subnet for a given GO device by making the OS replaces its default subnet with the proposed one. Fig. 3 shows the integration of these modules of ISNP with the Android software environment, where the application part of ISNP passes the proposed subnet to the OS part of ISNP using the standard Android APIs (i.e. using service discovery). Such integration facilitates the interaction between the two modules without breaking the application code, as no special APIs are required. Moreover, if it is not possible to modify the OS on a device, such a device would still be able to participate in ISNP using its application code. In this case the device would be able to propose a subnet and inform other devices about such subnet and any possible conflicts. However, such a device will not be able to change the subnet because the OS part of ISNP is not available. Thus, this device will not be able to assume a GO responsibility, but it can join a group as a GM.

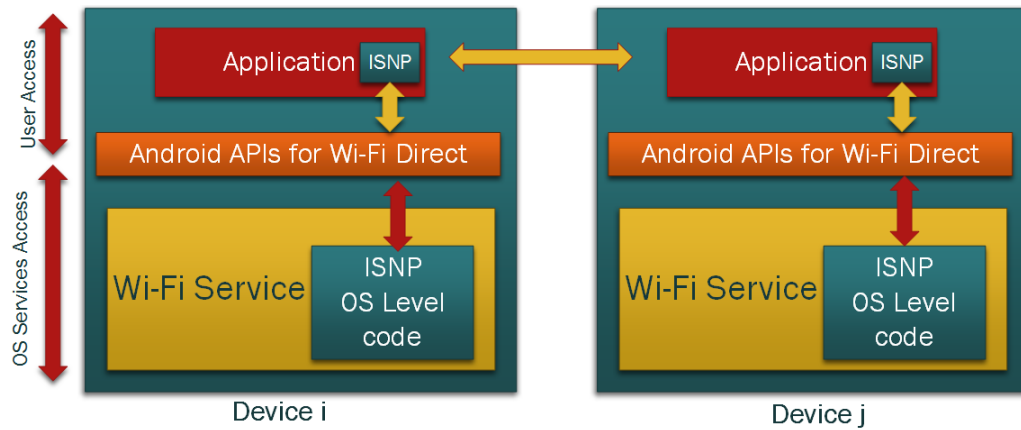


Figure 6-3 The integration between the two part of ISNP and Android.

For the proposed IP subnets, ISNP replaces the default subnet in Android, which is 192.168.49.0/24, by 10.X.Y.0/24 which enables flexibility in having IP subnets that

can range from 10.0.0.0 to 10.254.254.0. We discuss the two ISNP modules in the balance of this section.

6.2.2 Application-Level Module

ISNP utilizes the service discovery records that EMC uses for exchanging information between devices. ISNP introduces a modified version of the DeviceInfo record of EMC that was shown in Figure 5-2 by adding a “SUBNET” field as shown in Figure 6-4. This approach simplifies the IP subnet announcement process, as it is done at the same time when the devices exchange their information. The operation of the application level-module of ISNP is discussed in the balance of this section.

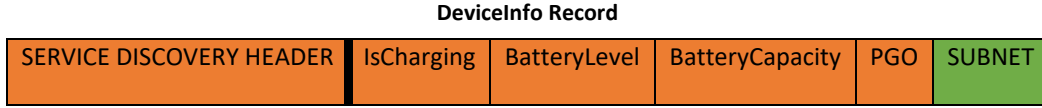


Figure 6-4 The new format of EMC's DeviceInfo record that is used by ISNP

6.2.2.1 The Operation of the Application-Level Module

As mentioned earlier, the subnet range chosen for ISNP is 10.X.Y.0. To reduce the length of the DeviceInfo record, we send only the “X.Y” part of the subnet. At the beginning of ISNP, each device D_i generates a random IP subnet, as explained in section 6.2.2.2, and stores it in the SUBNET field during the period of $T_{declareGO}$ that is mentioned previously in section 5.2.1 of EMC. As in EMC, devices executing ISNP continuously request services from each other each $T_{sendInterval}$ period; thus, D_i will receive the DeviceInfo record of other devices and provides its own in response to their inquiries. Upon receiving a response from D_k , D_i extracts the embedded DeviceInfo record and stores it. At this point, D_i checks whether the SUBNET field of any of the

collected DeviceInfo records is conflicting with its proposed one. In the presence of a conflict, D_i generates a new random subnet and updates its SUBNET field with it. $T_{declareGO}$ should be large enough to accommodate for the propagation of proposed subnets and to allow any conflict to be resolved.

Conflicts can also be detected by other devices. To elaborate, let us assume that a device D_2 falls within the communication range of two disjoint devices, D_1 and D_3 . As D_1 and D_3 cannot reach each other, if they propose the same subnet they will not know about it. Moreover, it can happen that D_1 and D_3 create their own group as well and they select D_2 as a relay for these two groups. In such a case, D_2 will be a PM in two conflicting groups, which is the same problem we are trying to solve. To make sure that this situation does not happen D_2 should inform D_1 and D_3 about the conflict. Thus, we add to the SUBNET field in the DeviceInfo record, in addition to the locally proposed subnet for the device, any conflicting subnets that a device detects between its neighbors. The final format for the SUBNET field is a comma separated values of different subnets (e.g. X1.Y1,X2.Y2, ...), where the first value is for the proposed subnet by the device and the rest are the detected conflicts between neighbors, if any.

Figure 6-5 illustrates the operation of the application-level module through an example. Three devices D_1 , D_2 , And D_3 are in the range of each other. D_1 proposed 201.23, D_2 proposed 63.56 and D_3 proposed 84.45. So, D_1 updates its SUBNET field to “201.23”, and D_2 and D_3 do the same. All of them will notice no conflict when receiving the DeviceInfo records of the others. Assume that D_4 which is in the range of D_3 and not D_2 , is also executing ISNP. D_4 happens to choose 63.56 as its subnet. D_3 in

this case will detect the conflict between D_2 and D_4 and will add this conflicting subnet to its SUBNET field to inform them about the situation. This means that the new SUBNET field for D_3 will be “84.45, 63.56”. Upon receiving the new DeviceInfo record from D_3 , both D_2 and D_4 propose new randomly-selected subnets.

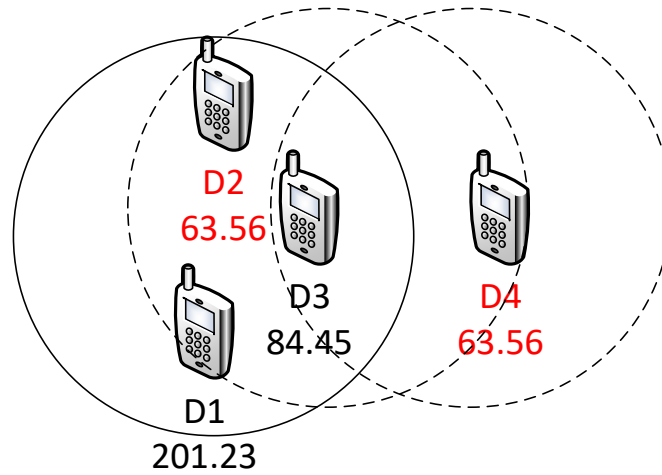


Figure 6-5 Resolving conflicts on ISNP.

6.2.2.2 Randomly Generating Subnets

To lower the probability of having two or more devices picking the same subset, ISNP uses the MAC address of the Wi-Fi transceiver, which is unique among devices, to define a seed for the random number generator of the individual devices. Basically, each device retrieves its MAC address and then performs a bit wise addition and shifting on the MAC address bytes to generate the seed. Since we have the subnet in the form “X.Y”, we generate X first then Y. The numbers X and Y are randomly generated in the range 0 to MaxX and 0 to MaxY respectively, where MaxX and MaxY are chosen based on the number of anticipated devices. After that we concatenate both

of them in a string “X.Y”. In the real world, some Wi-Fi routers have their default subnets assigned to 10.0.0.0, 10.0.1.0, 10.1.1.0, 10.1.1.0, 10.2.2.0, or 10.10.1.0. To avoid any potential conflicts with such Wi-Fi networks, we have chosen to avoid these addresses. A pseudo code for the application module of ISNP is shown in Figure 6-6.

```

1 Initialize IRMC ()
2   startTime = System.TimeNow
3   macAddress = RetrieveWifiMacAddress
4   seed = GenerateSeed ();
5   randomGen = NewRandom(seed)
6   subnet = GetProposedSubNet()
7   myDeviceInfoRec = New DeviceInfo Record
8   myDeviceInfoRec.SUBNET = subnet
9   Store myDeviceInfoRec as LocalService
10  Periodically, DiscoverNearbyServices()
11  While Sytem.TimeNow - startTime <= TdecLareGO do
12    If ServiceRespose Received from Neighbor Then
13      response = ServiceRespose
14      While CheckConflict(respose.deviceInfoRec.SUBNET) do
15        subnet = GetProposedSubNet
16        myDeviceInfoRec.SUBNET = subnet
17      End If
18      If CheckConflictInNeighbors() Then
19        myDeviceInfoRec.SUBNET = subnet + conflictedNeighborSubNet
20      End If
21    End While
22  GenerateSeed ()
23  seed = 0
24  For each Byte in macAddress do
25    seed = seed <<8 + macAddress[curByte]
26  End For
27  Return seed
28 GetProposedSubNet ()
29  sub1 = randomGen.getNext(3, MaxX)
30  If sub1 = 10 Then
31    sub1 = randomGen.getNext(3, MaxX)
32  End If
33  sub2 = randomGen.getNext(3, MaxY)
34  Return sub1 + "." + sub2
35 CheckConflict (SUBNET)
36  For each d_subnet in SUBNET do
37    If d_subnet == subnet Then
38      Return True
39    End If
40  End For
41 CheckConflictInNeighbors ()
42  For each Neighbor do
43    If Neighbor_i.subnet == Neighbor_j.subnet Then
44      conflictedNeighborSubNet += Neighbor_i.subnet
45    End If
46  End For

```

Figure 6-6 Pseudo code for the application part of ISNP

Given that we have $MaxX$ and $MaxY$ as the maximum allowed ranges for the subnets, ISNP will have $(MaxX + 1) \times (MaxY + 1)$ subnets to pick from. Let $P(S_i)$ be the probability of selecting a certain subnet. Thus,

$$P(S_i) = \frac{1}{(MaxX + 1) \times (MaxY + 1)}$$

6.2.3 OS-Level Module

This module of ISNP replaces certain parts of the Android OS to allow a device to use its proposed subnet instead of the default one. Let us first discuss the flow of Wi-Fi Direct commands that are issued from the app until getting executed. When a Wi-Fi Direct API is called from an application, the Android invokes a service called “wifi_service” which is responsible for translating such a request into a command for a low-level service called “wpa_supplicant”. *wpa_supplicant* then deals with the Wi-Fi drivers to perform the actual operation.

Tackling the problem of negotiating the IP subnets in the *wpa_supplicant* is possible; however, it will involve a datalink layer service in negotiating a network layer aspect and is thus not recommended. We have, therefore, decided to split the implementation of ISNP into an application and an OS parts. The OS module of ISNP resides in the *wifi_service* and intercepts any local services added by applications, which will contain a DeviceInfo record if the application-level module of ISNP is running. At the *wifi_service* module of Android, the local services are prepared and packed by the OS in a format that is understood by the *wpa_supplicant*. During such packing process, headers are added and the data is changed to a Hex string. The

interception of the local services by ISNP requires unpacking the records and decoding their fields. The unpacking process reverses all what the Android OS has done to create such a local service in order to obtain the original record that is created by the application level module. If a DeviceInfo record is found the OS module of ISNP proceeds to change the default IP address of the device to match the first entry in such SUBNET field, which is the subnet proposed by the application-level module for the current device. The remaining subnets in the SUBNET field, if found, are ignored as they are meant for informing other devices about conflicts. ISNP then changes the range of the DHCP server to fall in the range 10.X.Y.2 to 10.X.Y.254 for a given subnet of “X.Y”. After that the normal operation of exchanging such DeviceInfo record with other devices is continued.

For devices with unpatched versions of Android, the *wifi_service* will receive the DeviceInfo record and treat it as any regular service discovery record. Thus, it will exchange it with other devices nearby. The device will not attempt to inspect the record for the SUBNET field as it does not know about such a field. Therefore, the operations of the application part of ISNP would continue to run and the OS will not crash. However, the proposed subnet will be ignored by the OS of the unpatched device.

6.3 Implementation and Testing

For implementing the OS-level module of ISNP, we have made the required modifications to Android to enable dynamic IP subnet assignment. A popular custom version of Android, Lineage OS (previously Cyanogenmod), is used for implementing the modifications. We chose Lineage OS because of its support of wide range of

devices, which means that our modifications can be adopted easily by many devices. To implement the application-level module of ISNP, the application that we developed previously for EMC in AndroidStudio is modified to include ISNP functionality. Figure 6-7 shows screenshots of the app while it is running on two different devices.

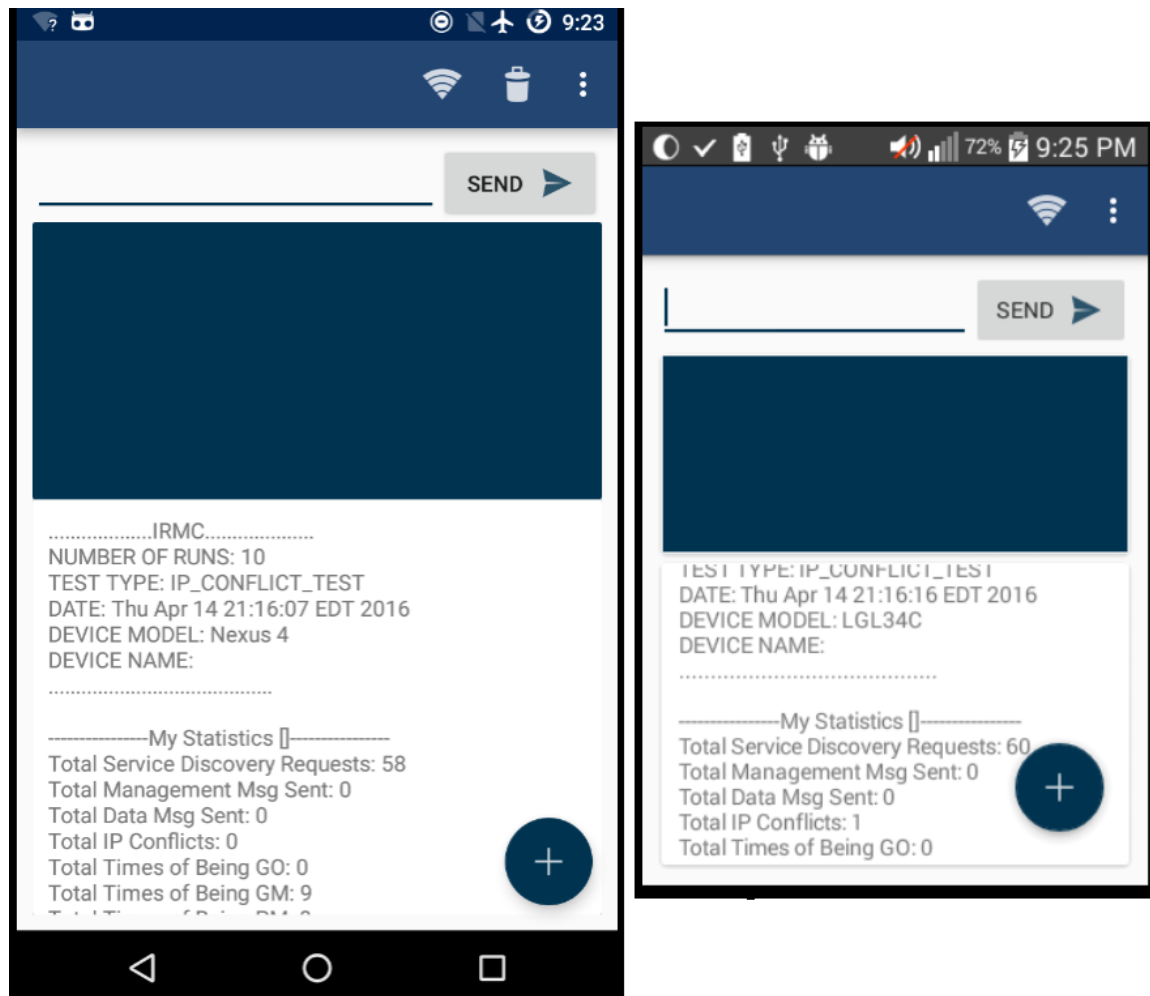


Figure 6-7 Screen shots of ISNP on Nexus4 and LG Optimus Fuel

The application is installed in seven devices with different specifications as shown in Table 6-1. All tests were run in our laboratory, where many Wi-Fi access points exist

in the building and use the same frequency band of Wi-Fi Direct; this reflects practical scenarios in terms of medium access contention.

Table 6-1 The specifications of the devices involved in testing

Count	Device	Android Version	RAM
1	Nexus 6	Lineage OS 6.0.1	3 GB
2	Nexus 4	Lineage OS 6.0.1	2 GB
2	Galaxy Tab 2	Stock 4.2.2	1 GB
2	LG Optimus Fuel	Stock 4.4	512 MB

The two parameters of EMC, $T_{sendInterval}$ and $T_{declareGO}$, need to be preconfigured to allow optimal operation of ISNP. Recall that $T_{sendInterval}$ is used to determine the period at which a device would ask other devices about their services (i.e. discover DeviceInfo records from nearby devices), and $T_{declareGO}$ denotes the time allowed for devices to negotiate their subnets before moving to the next step, which is handling the creation of groups. Thus, each device sends inquiries, or negotiates and resolves conflicts, $T_{declareGO}/T_{sendInterval}$ times to other devices on each run of ISNP. To set a proper value for $T_{sendInterval}$, we must first measure how fast is the response from sending a request until receiving the answer. Obviously, we do not need to send another request before completing the first one. Thus, we evaluated the response time in different configurations. To adjust $T_{declareGO}$, we need to get an estimate of the number of conflicts per run and then determine $T_{declareGO}$ to allow the devices to request DeviceInfo records (i.e. to negotiate and resolve conflicts) several times greater than the expected conflicts. An evaluation of such number of conflicts is conducted next, which give an insight of how to set $T_{declareGO}$.

6.3.1 Response Time Performance

In this experiment, we evaluate how fast ISNP is in reporting proposed subnets. We have varied the number of devices from 2 to 7 to assess the impact of the density of devices on the performance. In each case, we let the involved devices create a DeviceInfo record and store it. Then each device requested the DeviceInfo record of other devices. The time from sending the request until the replies of all devices in the experiment are received is denoted as the response time. For example, if we have 2 devices in the test, then each one of them will record its response time when one record is received. For seven devices, a device waits for receiving 6 responses before recording its response time. We ran the experiment 5 times for each configuration (i.e., number of involved devices) and recorded for each device its response time in each run and then calculated the average. Figure 6-8 and Figure 6-9 show the results.

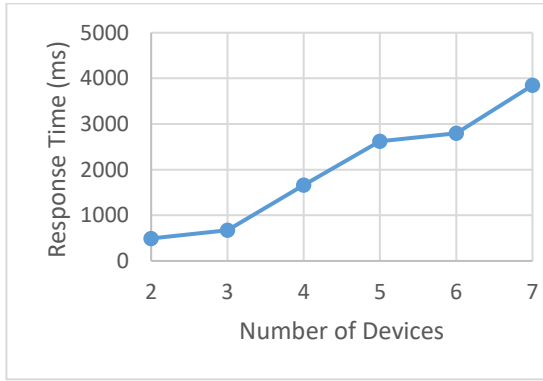


Figure 6-8 Average response time of ISNP with device count.

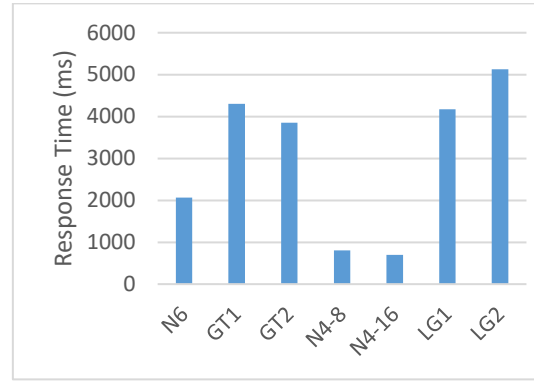


Figure 6-9 Average response time of ISNP per device.

In Figure 6-8, the average response time (in msec) is plotted against the number of devices. This figure shows that the response time is increasing almost linearly with the

number of devices. This is expected since the number of responses that a device should wait for grows linearly with the device population. Figure 6-9 shows the average response time per device. The interesting observation here is that the response time is dependent on the device itself. We can see from the figure that the response time for the older devices, Galaxy Tab (GT1 and GT2), and the LG Optimiums Fuel (LG1 and LG2) is quite high compared to the more powerful Nexus 4 (N4-8 and N4-16). However, the Nexus 6 (N6) is showing higher response time compared to the Nexus 4. This unexpected result indicates that the Murata Wi-Fi chipset in Nexus 4 is more optimized compared to the Broadcom chip on the Nexus 6. What worth noting also is that the Galaxy Tabs that show high response times are also having a Broadcom chip. We thus note that certain hardware is more optimized and can run Wi-Fi Direct operations more efficiently than the other. Based on the observed response time, we chose $T_{sendInterval} = 6$ seconds as a safe period for repeatedly asking other devices for their DeviceInfo records. Such choice is applied in the next experiment.

6.3.2 Subnet Conflict Evaluation

In this experiment, we opt to capture the relation between the number of devices and the number of possible conflicts in the range 10.X.Y.0 of subnets used in ISNP. We have fixed the number of devices to 7 and varied the allowed subnet range that a device can choose from by changing the parameters of MaxX and MaxY. Although it is possible to change the values of MaxX and MaxY individually, we chose to set both to the same value. Also, we drew one random number only and applied it to both parts of the proposed subnet, X and Y. We have started with setting MaxX and MaxY to 7 and

increased this number to 50 then 100 and so on until 350. With such configuration, we get 8, 51, 101, ..., 351 subnets respectively. We ran this experiment 10 times for each subnet range. With each subnet range, we recorded the number of conflicts that ISNP detected and resolved in each device. The total number of conflicts per range is plotted in Figure 6-10, while the average number of resolved conflicts per device during each run of the experiment is shown in Figure 6-11.

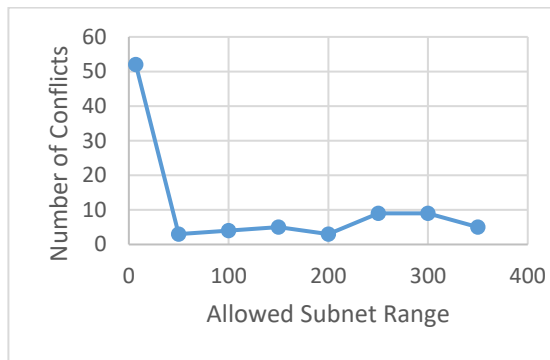


Figure 6-10 Average response time of ISNP with device count.

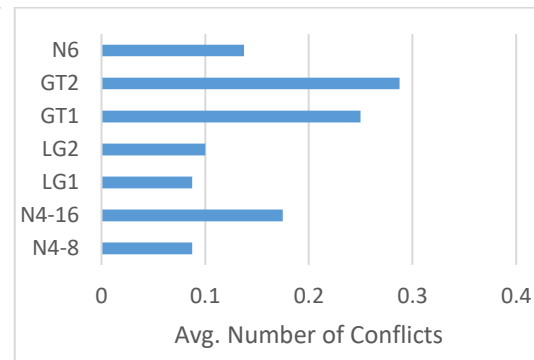


Figure 6-11 Average response time of ISNP per device.

It is noted from Figure 6-10 that the number of conflicts is very high when only eight subnets are allowed, as 52 conflicted have been reported during the 10 runs of ISNP in such range. The number of conflicts have decreased drastically for the other ranges and stabilized below 10; this is very much expected due to the larger set of subnet addresses. Meanwhile Figure 6-11 shows that in the worst-case a device has experienced about 0.2875 conflicts per run for all experiments. This is quite low, especially when considering the limited subnet ranges used in the experiment. It worth noting that managing the parameters of ISNP, MaxX and MaxY, we could reach up to

64511 possible subnets. Based on these results, we can choose the value of the parameter $T_{declareGO}$ to be at least double the value of $T_{sendInterval}$ (i.e. $T_{declareGO} > 2 T_{sendInterval}$) to allow the conflicts to be detected in a timely manner.

6.3.3 Integration with EMC

We have tested also the effect of integrating ISNP with our multi-grouping protocol, EMC, on the final network topology. Three of our test devices are patched to run ISNP OS module. A complete run of the application with ISNP integrated with EMC is performed where the devices start negotiating subnets using ISNP and then create groups and share data using EMC. In this case, only the patched devices were allowed to create groups. Upon creating the groups, we examined the IP addresses of the devices and confirmed that the IP subnet range of each group was the same as proposed, which confirms successful integration between the application and OS modules of ISNP. In addition, successful bi-directional communications between the groups through EMC were possible.

6.4 Conclusion

In this chapter, we have presented ISNP, a protocol that facilitates the multi-group data sharing in Wi-Fi Direct by providing a mean for assigning different subnets for adjacent groups. ISNP integrates with EMC to provide a connectionless negotiation of IP subnet between Wi-Fi Direct devices using the service discovery mechanism. Each device participating in ISNP selects a random IP subnet and share it with nearby devices. If a conflict in the chosen subnet is detected a new IP subnet is picked. To facilitate the integration with contemporary portable devices, ISNP is composed of two

parts, one at the application level and another one at the Android OS level. The ISNP flexibility allows devices that have unmodified Android OS to still participate in ISNP, with the restriction that they should not be selected as GOs. An Android application is used to demonstrate the ability of ISNP to efficiently assign different IP subnets to different devices. The performance results have shown that while ISNP is robust in terms of avoiding conflicting IP subnets, the latency of the IP subset agreement varies significantly based on the processor capability and Wi-Fi Direct transceiver used on the involved devices.

Chapter 7: Simulation Experiments

In this chapter, we validate the scalability and performance of our proposed protocols through simulation. Although we implemented our protocols on typical Android phones, the simulation will enable studying the performance when many devices are involved. To the best of our knowledge, there are no available simulator for Wi-Fi Direct that can be used to model the interaction between a large number of devices. Thus, we opted to implement our own simulation framework for Wi-Fi Direct; we made it also freely available on GitHub³ for other researchers to benefit from it. This chapter discusses the simulator, the setup of the experiments, the performance metrics, and the obtained results.

7.1 Building a Simulator for Wi-Fi Direct

Although quite a few network simulators are available, none of them support Wi-Fi Direct. Thus, we needed to build our own simulator for Wi-Fi Direct by extending one of the existing tools. Two criteria have been applied when selecting which of the existing simulation tools to extend. First, the tool should be well structured and support the implementation of the ANSI seven-layer protocol stack model in order to mimic the exact behavior of devices in practice, e.g., signal propagation, collisions, layers of the TCP/IP model, etc. The second criterion is the availability of the source code so that modifications can be made and new modules can be integrated. Details are provided in the following subsections.

³ <https://github.com/ashahin1/inet>

7.1.1 Tools Used for the Simulator

Our simulator is based on OMNeT++ [83] and the INET Framework [84]. OMNeT++ is a discrete event simulator that is written in C++. It has a very powerful simulation kernel that contains tools for starting and stopping the simulation, defining and configuring modules, performing communications between modules, performing statistical operations, and recording and displaying measurements. The INET Framework (or INET for short) is a model suite for wired, wireless and mobile networks that is built on top of OMNeT++ (i.e., it is written using the simulation primitives provided by OMNeT++). INET has been used to implement several wireless networks in OMNeT++, such as Bluetooth (802.15.4), and Wi-Fi (802.11), with fine-grained details, i.e., signal propagation, modulation of radio signals, power consumption, the MAC layer, the network layer, the transport layer, and the application layer. INET also has an extendable architecture that facilitates adding and/or replacing existing modules. Despite having support for 802.11 networks, there is no support for Wi-Fi Direct in INET. Thus, we used the 802.11 modules of INET as a base for adding the Wi-Fi Direct functionality. We have also utilized Google OR-Tools [85] for the Hungarian (Munkres) algorithm implementation and for checking the connectivity status (number of connected components) in the resultant networks. Next, we will explain the implementation in detail.

7.1.2 Implementing the Simulator

Wi-Fi Direct operations on real devices are handled by the Wi-Fi network interface card (NIC). This NIC is responsible for both infrastructure Wi-Fi and Wi-Fi Direct

connections. To allow these two different connections to concurrently exist, a virtual interface for each of them is created, e.g., “wlan0” and “p2p”. To allow the NIC to have concurrent operations from both virtual interfaces, two separate MAC entities are used, each operates on a different channel. One of the MAC entities is used to handle the infrastructure Wi-Fi operation of “wlan0” and track the status of the connection through a state machine. The other entity handles the operation of Wi-Fi Direct “p2p” with a totally different state machine. A device with such configuration can work as a client in a WLAN and a GO/GM at the same time.

As a simulation framework for wired and wireless networks, INET has implemented the Wi-Fi (802.11) operation with fine-grained details, using the primitives of OMNet++. The networking aspects has been implemented as modules that provide functionalities related to each layer. For example, the physical layer is implemented using bit manipulation, packet handling, and radio modules. Module composition is pursues to implement complex functionality; for example a NIC is composed of management, MAC, and radio modules. In INET, an 802.11 NIC can be configured as STATION, ACCESSPOINT, or AD-HOC, where each configuration has its related MAC aspects. A device is modeled as a module that is composed of applications, transport layer (TCP, UDP), network layer (IPv4, IPv6), and NICs.

We have utilized the modules of INET to create a device that support Wi-Fi Direct operations. For Wi-Fi Direct a device should be able to do service discovery, and act as a GO and as a GM.

The service discovery operations are handled in a connection less manner and have broadcast nature. These operations can be performed without forming a group. We have modelled such operations using an 802.11 NIC configured as AD-HOC, given that we allow the devices to broadcast their data. The GO role is handled by creating a software access point (SAP). Using Wi-Fi Direct, devices can connect to such SAP using a simple invitation/acceptance mechanism. All devices (Legacy and Wi-Fi Direct enabled) can connect using the SSID and Key. In both cases, the connected device will be regarded as GM. Therefore, we have implemented the functionality of the GO using an 802.11 NIC configured as ACCESSPOINT. Likewise, the GM role is modeled using an 802.11 NIC configured as STATION. Such GM then connects to the GO using the SSID and the Key. To allow a device to have infrastructure Wi-Fi support besides the Wi-Fi Direct support, we have added another 802.11 NIC and configured it as STATION.

Based on the above discussion, we have built a host that supports Wi-Fi Direct using 5 NICs where one of them is just a loopback interface that is required by INET. The remaining four are two 802.11 NICs (STATION), one 802.11 NIC (AD-HOC), and one 802.11 NIC (ACCESSPOINT). To mimic the behavior of real NICs that support Wi-Fi Direct by using two virtual interfaces, we have added logic to selectively disable unwanted NICs so that, at most two wireless NICs will be active at the same time. For example, a device entering service discovery would have the ad-hoc NIC active and the other 3 wireless NICs disabled. If that device becomes a GO, its access point NIC will be activated. At the end of the service discovery period, the ad-hoc NIC will be

disabled. Likewise, a GM device will have one of the station NICs active. Since the assignment of GMs marks the end of the service discovery period, the ad-hoc NIC for GMs will be disabled when they start their role. If a GM happen to serve as a PM, then its other station NIC is activated. Figure 7-1 show the design of the Wi-Fi Direct host that we utilized in the simulator.

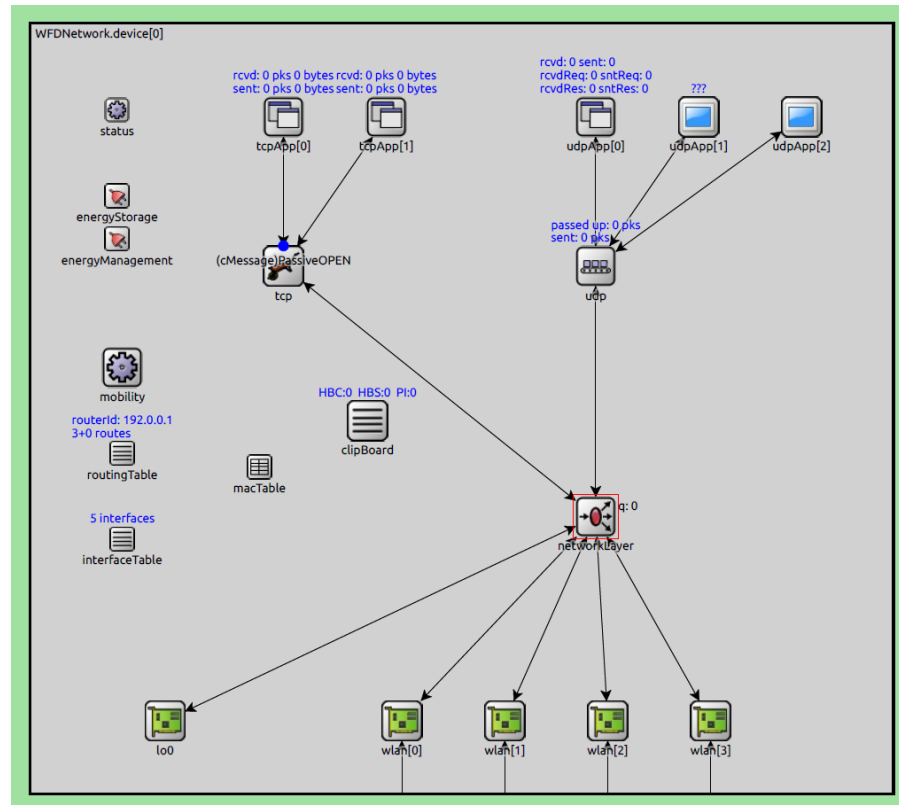


Figure 7-1 The internal design of a Wi-Fi Direct Host.

For implementing the service discovery operations, we have built a UDP application in the designed host that manages the distribution of service discovery records, by broadcasting them to reachable neighbors. Then we have utilized this application to add ADS, EMC, and ISNP. Two other UDP applications, a DHCP client and a DHCP

server, are used to aid in finalizing the process of joining groups; a GM needs an IP address to communicate with its GO. Each device has these two UDP applications, but only one will be active at a time depending on the role of the device. For GOs, the DHCP server configuration is updated to reflect the subnet negotiated by ISNP. The GMs use their DHCP client application to request an IP address lease. For the intra-group communications, we have designed two TCP applications, a server and a client, where one of them will be active based on the role of the device. We then added ELN to these two applications to be able to manage the group. Thus, a GO device utilizes the TCP server to send its heartbeat messages and a GM device utilizes the TCP client to receive them. The part of EMC that is responsible for the PM assignments is utilizing these two TCP applications as well.

Our simulator design fully support the communication protocol stack. For example, there are SYN and FIN commands to open TCP connections. There are also server and client sockets. A message sent from the application is encapsulated as it passes through the different layers and de-capsulated at the other end. The station connects to the access point using the 802.11 procedures defined in the standard.

7.2 Experiment Setup

We have done two different types of experiments, one to assess the protocols performance, which is explained in section 7.4.1, and another in section 7.4.2 to study the effect of parameters on our protocols. We set the Wi-Fi transceiver to 802.11g, which gives a bit rate of 54Mbps. The transmission power is set to 0.9mW, which gives a range close to 802.11g range of almost 150m. The path loss model is set to free space.

We used two different type of topologies, a static grid, and a stationary connected graph. Table 7-1 summarizes all the assumptions that we used for all the parameters, unless otherwise stated. Every experiment is repeated 30 times with different seeds and the average of them is obtained and plotted. We observed that with 90% confidence level, the simulation results stay within 5% of the sample mean.

Table 7-1 List of the paramters used for the simulation

Parameter	Protocol Testing Experiments	Parameters Effects Experiments
Area	1Km \times 1Km	500m \times 500m
Devices Count	50 to 500 step 50	100
Topology	Static Grid + Stationary Connected Graph	
Wi-Fi Transceiver	802.11g	
Transmission Power	0.9mW	
Nominal Battery Capacity	4J	5J
Initial Battery Capacity	Rand (2.0, 4.0)J	Rand (2.5, 5.0)J
$T_{sendInterval}$	2s	1s
$T_{declareGO}$	6s	4s
$T_{selectGO}$	4s	2s
$T_{HeartBeatGM}$	1s	
$T_{HeartBeatGO}$	3s	
$T_{pxAssignment}$	4s	
$MaxX, MaxY$	254	

We implemented EMC with ISNP. As we mentioned previously on Chapter 5, EMC is utilizing ADS for service discovery operations and ELN for intra-group management.

Thus, EMC has the implementation of both ADS and ELN embedded and all of their parameters are already used. For that reason, we focused on simulating EMC, which would also capture the performance of ADS and ELN as well. In addition, integrating ISNP with EMC enables the implementation of the complete P2P solution, which we refer to as Integrated EMC. For testing the performance of the Integrated EMC, we focused on two areas: 1) the GO declaration criteria 2) the PM assignment method.

For the GO declaration criteria, we compared the method used by the Integrated EMC, which we call GEMC, with two baseline criteria. The first baseline that we call GBAT is a method that we proposed in [80], where the battery information is used; however, it does not take into consideration the orphaned members problem that we mentioned in section 5.1.1.2. The second baseline is a random declaration of GO that we refer to as GRND, where each device randomly decides on being a GO regardless of the other devices. When comparing the performance of different GO declaration criteria, we fixed everything else to the defaults of the Integrated EMC.

The PM assignment method of the Integrated EMC, which we name MUNK, is tested against two other baseline methods. The first we name FRST, where the GO selects for nearby groups the first available member that can reach it. The second is a random selection, where the GO randomly selects for a certain group one of the members that can reach it; a method that we call PRND. The remaining operations of the Integrated EMC are kept the same.

For the Integrated EMC, the time $T_{decalreGO} + T_{selectGO}$ defines the period where the service discovery operations are performed, i.e. exchanging service discovery requests

and responses. The group management operations begin when the service discovery phase ends, and last for $T_{pxAssignment} + T_{teardown}$, where the GMs exchange heartbeat messages with the GO. We set the simulation time for all experiments to allow one full application cycle of the Integrated EMC, i.e $T_{simTime} = T_{decalreGO} + T_{selectGO} + T_{pxAssignment} + T_{teardown}$.

7.3 Performance Metrics

The performance metrics that we are interested in to assess the performance of our work are connectivity, response time, protocol overhead, power consumption, and subnet conflicts. Table 7-2 summarizes the performance metrics and how they are related to our protocols. Next, we will explain these metrics on more details.

Table 7-2 The relation between the different performance metrics and out protocols.

Metric	Related Measurements in the simulation	Affects/Affected By			
		ELN	ADS	EMC	ISNP
Connectivity	GO Count		×	×	
	GM Count		×	×	
	PM Count	×	×	×	
	Orphaned Count		×	×	
	CC Count	×	×	×	
Response Time	Service Discovery Request to Response Delay		×	×	×
	Management End to End Delay	×		×	
Protocol Overhead	Total Service Discovery Messages		×	×	×
	Total Management Messages	×		×	
Power Consumption	Total Consumed Power	×	×	×	

Subnet Conflicts	Resolved Conflicts		×		×
	Remaining Conflicts		×	×	×

7.3.1 Connectivity

This metric measures the ability of our protocols for forming connected P2P networks. For that we track the number of group owners (GO), group members (GM), proxy members (PM), orphaned members (Orph), and connected components (CC). A typical GO will consume more power than other devices, thus the number of GOs should be reduced to reduce overall power consumption; however, not having enough GOs to cover the area means disconnected network. In addition, the number of GMs should be large enough to allow for more choices when selecting PMs. The number of PMs is directly affecting the final connectivity of the network. Having fewer PMs compared to the number of groups means disjoint network components, as there will be not be enough number of proxies to connect the groups. The number of connected components should reach one to have a totally connected network.

Connectivity is directly related to how EMC creates groups and connects them using proxy members. In addition, the ADS parameters affect EMC decisions in creating groups, as they determine how the exchange of DeviceInfo and SAP records is performed. Moreover, ELN parameters that define the way of exchanging the management information also affect the proxy member selection.

7.3.2 Response Time

For response time, we are measuring how fast different data exchange operation can take place. The response time is also a key in tuning several parameters on our protocols. We are interested in the speed of exchanging service records, which affects ADS and consequently EMC and ISNP. Thus, we record the delay from sending a service discovery request and then receiving a response. We should tune the length of the service discovery period to accommodate for such delay. We are also interested in the speed of exchanging management data, which is related to ELN and EMC. Thus, we measure the end to end delay between sending a management message and then receiving it by the other party. Having a knowledge about this delay gives a hint on how the proxy selection period should last.

7.3.3 Messaging Overhead

The overhead is measured in terms of the number of messages that are exchanged during the protocol operation. Of course, a larger number means more bandwidth being consumed and more power consumption. We are interested in measuring the number of service discovery messages (sent and received), which is related to the ADS parameters and consequently EMC. In addition, we are considering the number of management messages (sent and received) that are affected by ELN parameters as well as EMC.

7.3.4 Power Consumption

The power consumption is crucial as it determines the lifetime of the devices. We are interested in decreasing power consumption. The choice of ADS, EMC, and ELN

parameters affect the power consumption. For example, increasing the frequency of requesting service discovery records leads to more power consumption.

7.3.5 Subnet Conflicts

The subnet conflicts refer to how many groups are sharing the same subnet. This is a measure of ISNP performance, where we are interested in eliminating the possibility of having conflicts. ADS and EMC parameters that are related to service discovery could affect such metric. The ability of having successful communication between groups at the level of the transport layer depends on this metric. We are recording in the simulation the number of resolved conflicts and the number of remaining conflicts, which should be zero to indicate the success of ISNP in resolving all the detected conflicts.

7.4 *Simulation Results*

As we pointed out earlier, we have two sets of experiments; the first is to compare the Integrated EMC with several baselines. Mainly, we test the GO selection criteria and the PM assignment method. The second set of experiments is for capturing the effect of changing various parameters on our proposed protocols.

7.4.1 Performance of Integrated EMC

The performance of the Integrated EMC is validated to assess its ability to create connected topologies while reducing the power consumption, overhead, and response time. Two different deployment types (topologies) are used for these experiments, the static grid, and the stationary connected graph. For the static grid configuration, we

divide the area into cells whose count matches the number of devices and placed one device into each cell; thus, we have equal spaces between the devices. Figure 7-2 shows an example of such deployment. In the stationary connected graph, we deploy the devices randomly in the area taking into consideration that each device should have at least one reachable neighbor. To do this, we deploy one device at a time and choose a random position for them, within the deployment area. For each deployed device, we check if there are other devices within its transmission range. If there are no devices within that range a new position is chosen and we repeat the same procedure until we find a suitable position. Figure 7-3 shows an example of this deployment type.

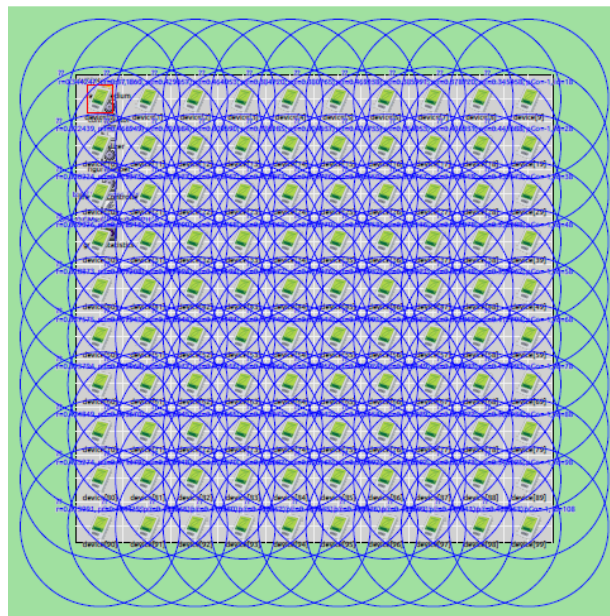


Figure 7-2 An example of the static grid deployment

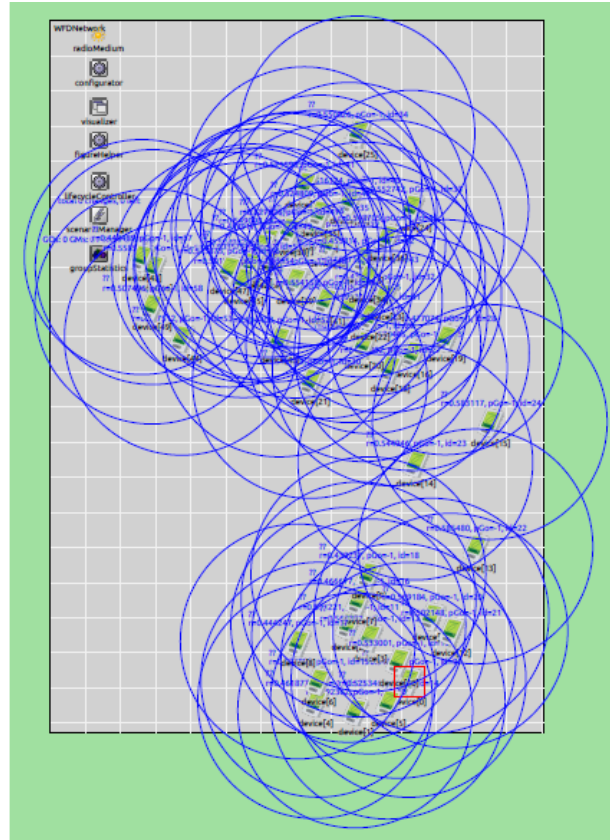


Figure 7-3 An example of the stationary connected graph deployment

7.4.1.1 GO Declaration Criteria

We first compare the performance of the GO selection criteria, GEMC, of the integrated EMC, against the GBAT and the GRND baselines. For this comparison, we changed only the part where the GOs declaration takes place in integrated EMC and let all other operation of the protocol to its defaults (ISNP for subnet conflict resolving, and MUNK for PM assignment). The GO declaration is crucial in determining the connectivity of the final peer-to-peer network. Covering the area with the least number of groups is desirable. The locations of the GOs also matters, as they may not be distributed evenly, which may affect the communication and the data exchange. In

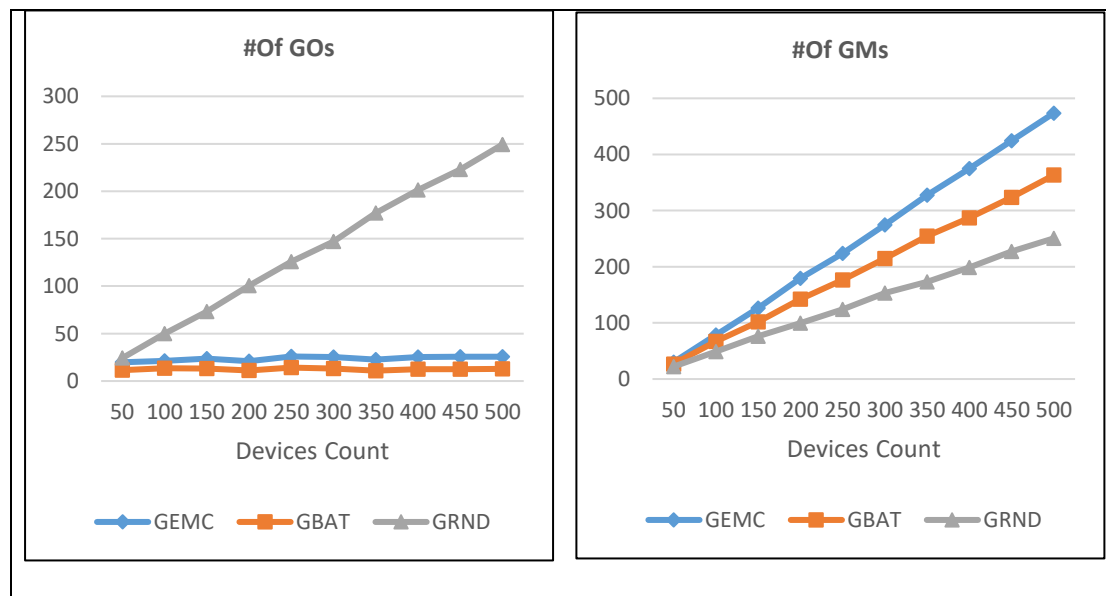
addition, having enough members in each group makes it highly possible to have enough PMs between a group and all its neighbors, which is a key in determining connectivity.

7.4.1.1.1 Static Grid

Figure 7-4 shows the resultant connectivity due to changing the GO declaration criteria in different device densities. The number of GOs should stay the same regardless the number of devices if the area stays the same. What we notice from the graphs is that GEMC is providing sufficient number of GOs to cover the area compared to GBAT and GRND, and yielding GO count that is not affected by the device density. GRND assignment for GOs is the worst, since there is no coordination between devices on who covers which region. We could see that the number of GOs in GRND is going up with density. For GBAT, the number of GOs is not affected much with the density of devices; however, it is not sufficient to give the best coverage as we will see next. The number of GMs is related to the number of GOs and orphaned devices. No orphaned devices were produced in case of GEMC on nearly all densities, thus all devices that are not serving as GO role have assumed a GM role, which could be noticed in the figure as well. Nearly the same is happening to the GRND, as the orphaned count is negligible. However, because there are too many GOs in case of GRND, the number of GMs becomes the lowest amongst the three approaches for all densities. The GBAT is producing too many orphaned devices, because it does not share the proposed GO with the nearby devices. Leaving devices out all groups affects the number of GMs, as seen in the figure. For the PM count, GEMC is in the middle between GRND and

GBAT for most densities. The GM and PM counts are correlated with the number of GOs, as we need more PMs to accommodate for the increased number of GOs.

The number of connected components is showing that both GRND and GBAT have many disjoint segments; and that number is increasing with the growth in the device population. For GRND, despite having many PMs, their count is not sufficient to cover the very large number of GOs, thus the number of connected components has increased. GBAT yields many orphaned devices, which causes the connectivity to suffer by having many disjoint nodes. GEMC, on the other side, has managed to keep the number of connected components low for all densities. Thus, in terms of connectivity, GEMC can produce sufficient number of GOs to cover the area, manage the number of GMs and PMs, avoid producing orphaned members, and obtain the best-connected network in all densities compared to GBAT and GRND.



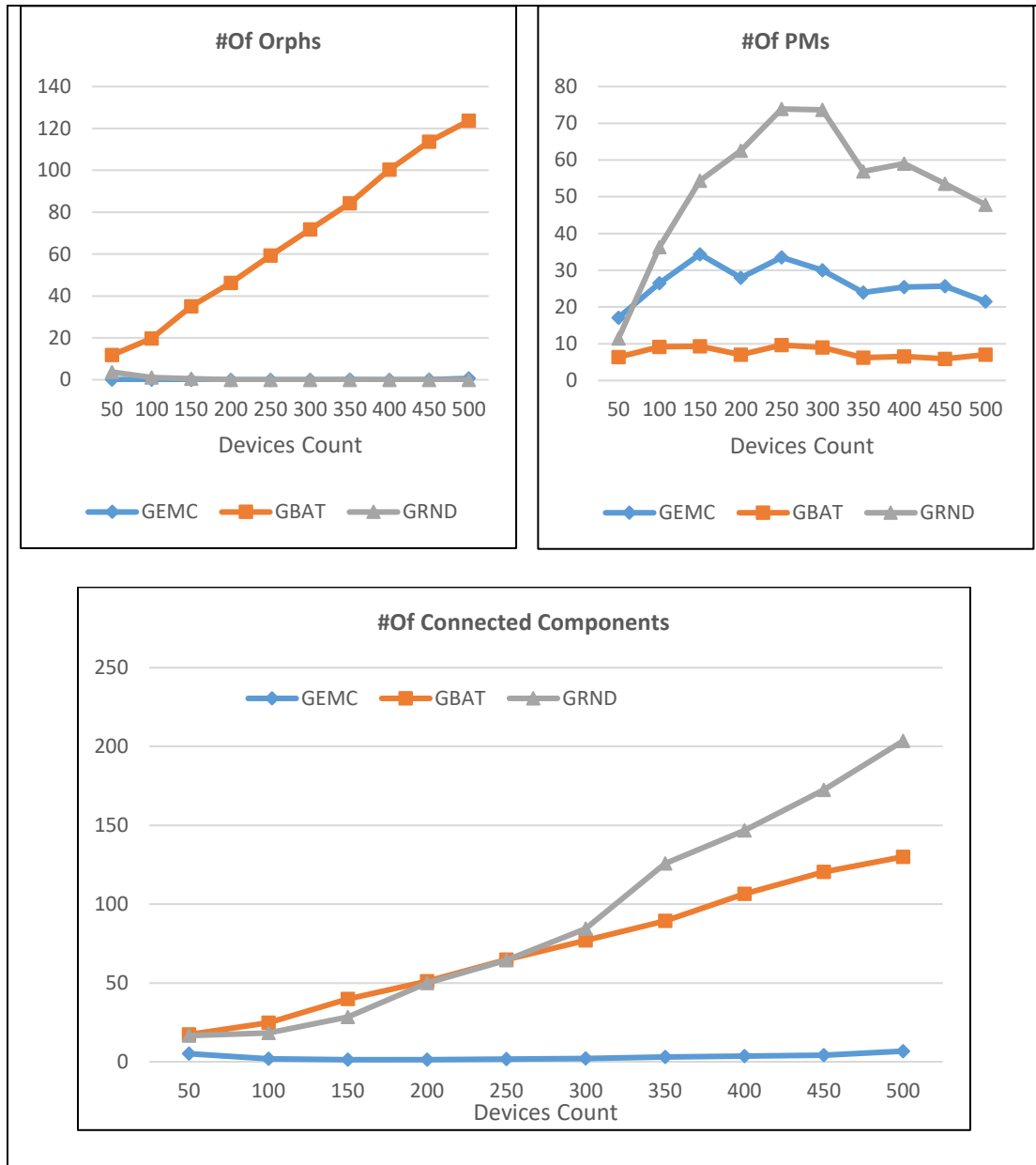


Figure 7-4 The connectivity results from GEMC, GBAT, and GRND in case of Static Grid topology

Figure 7-5 shows that the response time in the service discovery period grows with the number of devices, which is due to the increasing contention on the channel. GRND has the worst values due to having too many GOs contending on sending their SAP records. GEMC and GBAT perform nearly the same in terms of response time in the

service discovery period. Regarding the management messages delays, we notice that it also increases with device density. The GBAT is getting the worst delays here, because it has poor decisions in defining the group boundaries. GEMC outperforms GBAT and shows ability to evenly distribute the groups. GRND, however, appears to have less delays, but this is due to having very small boundaries for its groups, as the number of GOs is large.

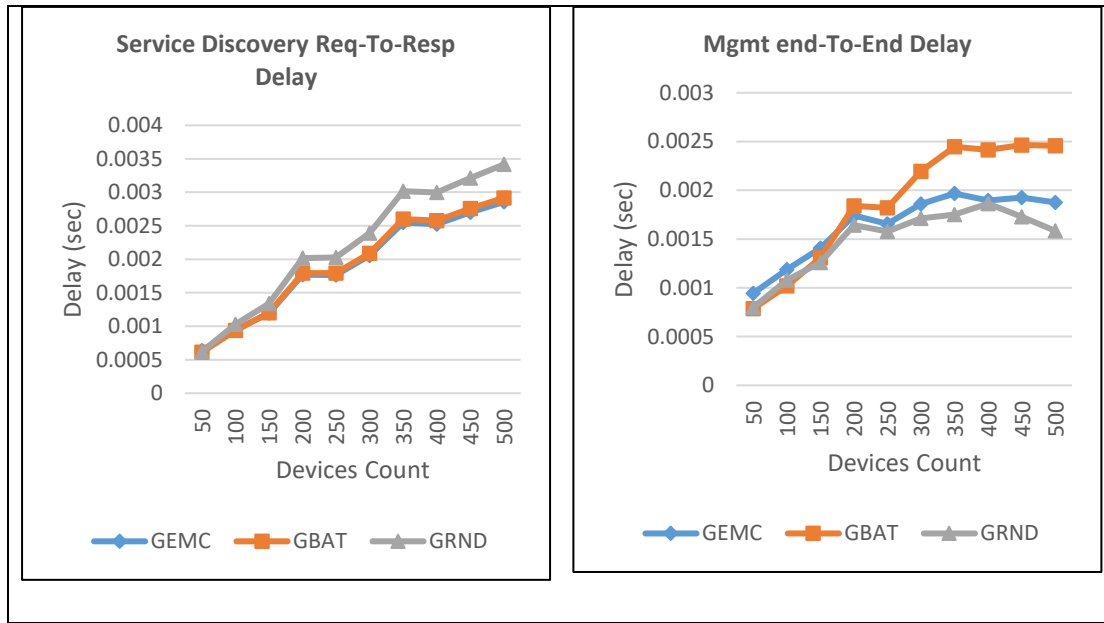


Figure 7-5 The response time results from GEMC, GBAT, and GRND in case of Static Grid topology

From Figure 7-6, we notice that the service discovery overhead for all approaches is increasing with density. However, GRND grows at the highest rate, due to the increased number of GOs. Increasing the number of GOs means increasing the exchanged SAP records between devices, thus increasing the number of service discovery messages. GEMC and GBAT need less messages than GRND, due to the designation of fewer GOs. The management messages are decreasing with density, which is due to the

interference from other devices. The management messages are sent using TCP, thus if there is an error due to interference the message will be discarded. What is recorded by this graph is the number of correctly received messages. GEMC is producing the highest number, which is due to the correct connectivity decision that lead to reducing the interference effect. The other two approaches are giving lower number of messages due to their inappropriate connectivity choices that signify the interference effects.

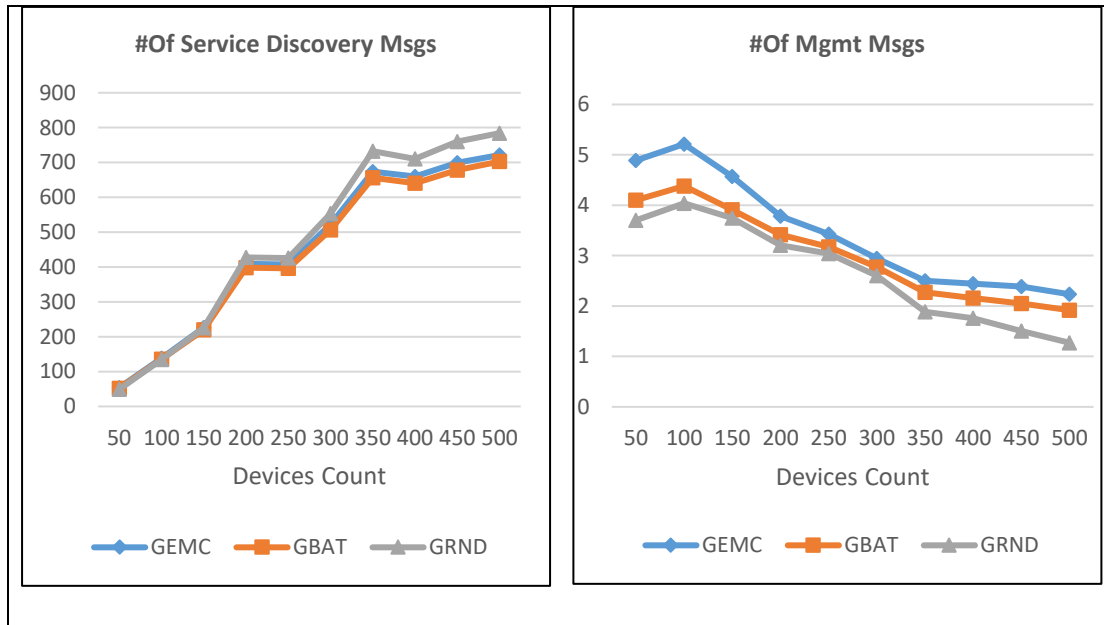


Figure 7-6 The overhead results from GEMC, GBAT, and GRND in case of Static Grid topology

The power consumption is shown in Figure 7-7, where we could notice that all the approaches consume more power when the density of devices increases. GRND gives the worst power consumption, due to having large number of GOs, which causes the exchange of more messages. GBAT is showing the best power consumption, but this is due to having many orphaned nodes that are not participating in any messages exchange. GEMC sets in the middle between the two in terms of power consumption.

If we combined that with the superior connectivity that we get from GEMC, we conclude GEMC is giving the best balance between connectivity and power consumption.

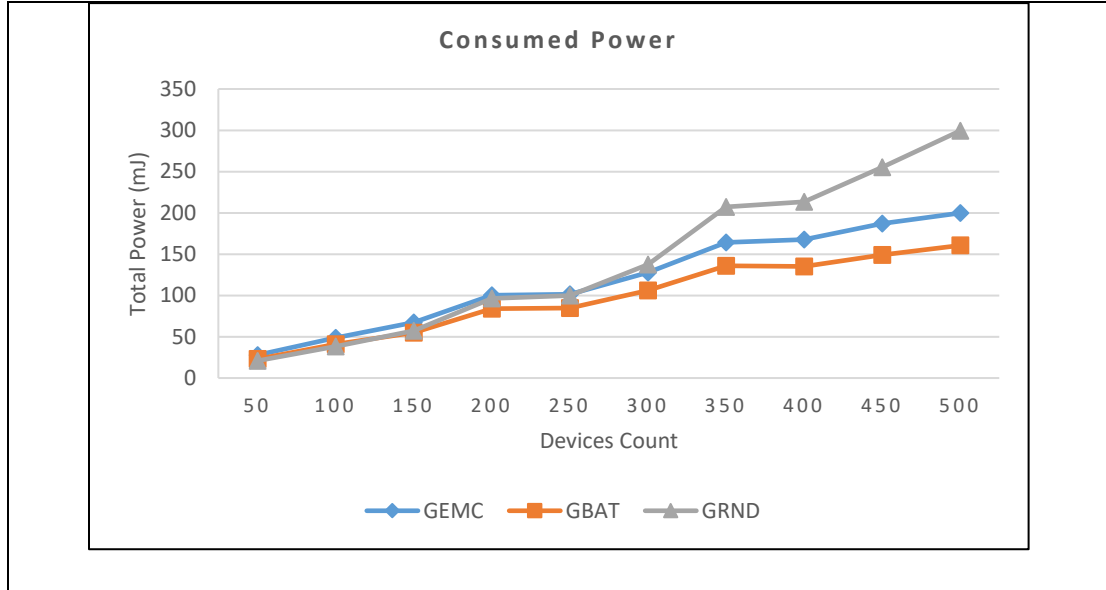


Figure 7-7 The power consumption results from GEMC, GBAT, and GRND in case of Static Grid topology

We also like to note that the number of detected and resolved subnet conflicts was negligible in all cases.

From this experiment, we conclude that GEMC is having the best balance between connectivity, message overhead, and power consumption in case of static grid topology. It also can adapt to the increased number of devices better than the other approaches.

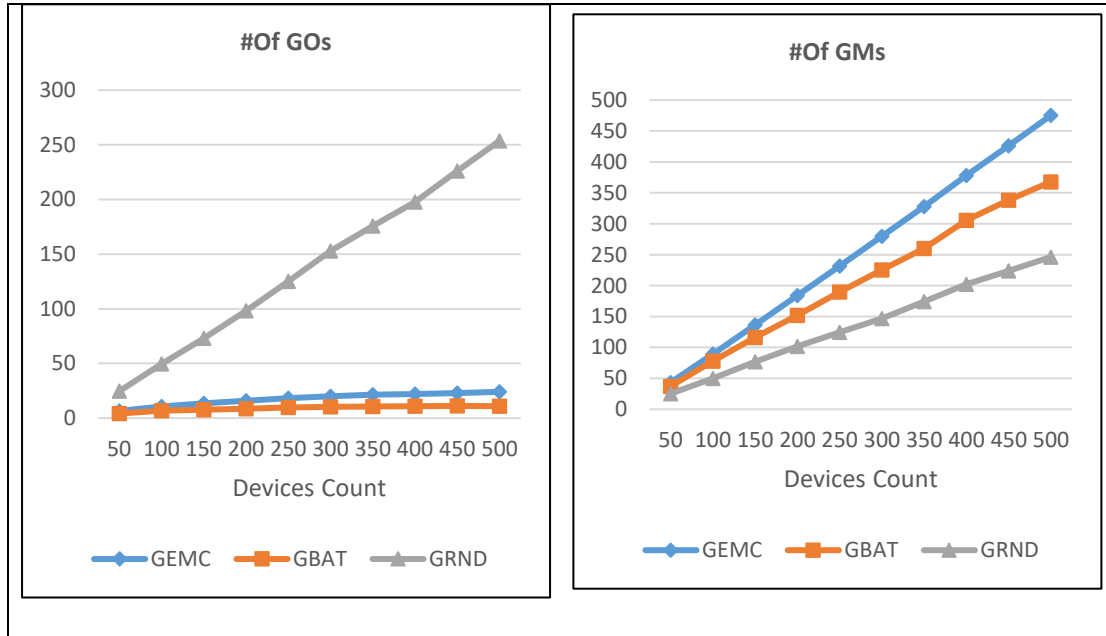
7.4.1.1.2 Stationary Connected Graph

In Figure 7-8 we show the resultant P2P device connectivity due to changing the GO declaration criteria in different device densities. GEMC managed to cover the area with enough GOs in all densities compared to GBAT and GRND. The number of GOs in

GEMC does not change with density, given the fixed area, which is a good sign of having a proper GO negotiation. The number of GOs produced by GBAT is not changing much, also, with the change in density, but GBAT does not yield the best coverage. The worst behavior is shown in case of GRND, where the number of GOs is increasing proportional with density, which is a result of randomly selecting the GO role independent of the other devices. GEMC and GRND can avoid producing orphaned members in all densities. However, GBAT suffers from having many orphaned members.

The number of GMs for the three approaches are analogical to the GO and Orphans count. We can notice that GRND has the lowest number of GMs in all densities; there are already many devices assumed the GO role, which means the remaining number of devices is already low. For all densities, GBAT has lower values of GMs compared to GEMC because GBAT has many orphan members. GEMC is best in terms of the GMs count, which would give more room for selecting PMs that could cover the network sufficiently. GRND designates more PMs with increased device density; however, the PM count is not enough to interconnect the formed groups as indicated by the number of connected components plot. GBAT is assigning the PM role to fewer devices compared to GEMC, a result that we can correlate with the number of GOs and GMs for both approaches. The number of connected components is best with GEMC, because of having the correct balance between GO, GM, and PM counts. The number of connected components is bad in both GRND and GBAT for all densities. For GRND, the PM count is not sufficient to cover the number of GOs, thus the number of

connected components increases. The orphaned members produced by GBAT cause the connectivity to suffer by having many disjoint parts. GEMC, on the other side, has managed to keep the number of connected components low for all densities. Compared with the static grid experiments discussed earlier, we conclude that the change of topology did not affect the results much.



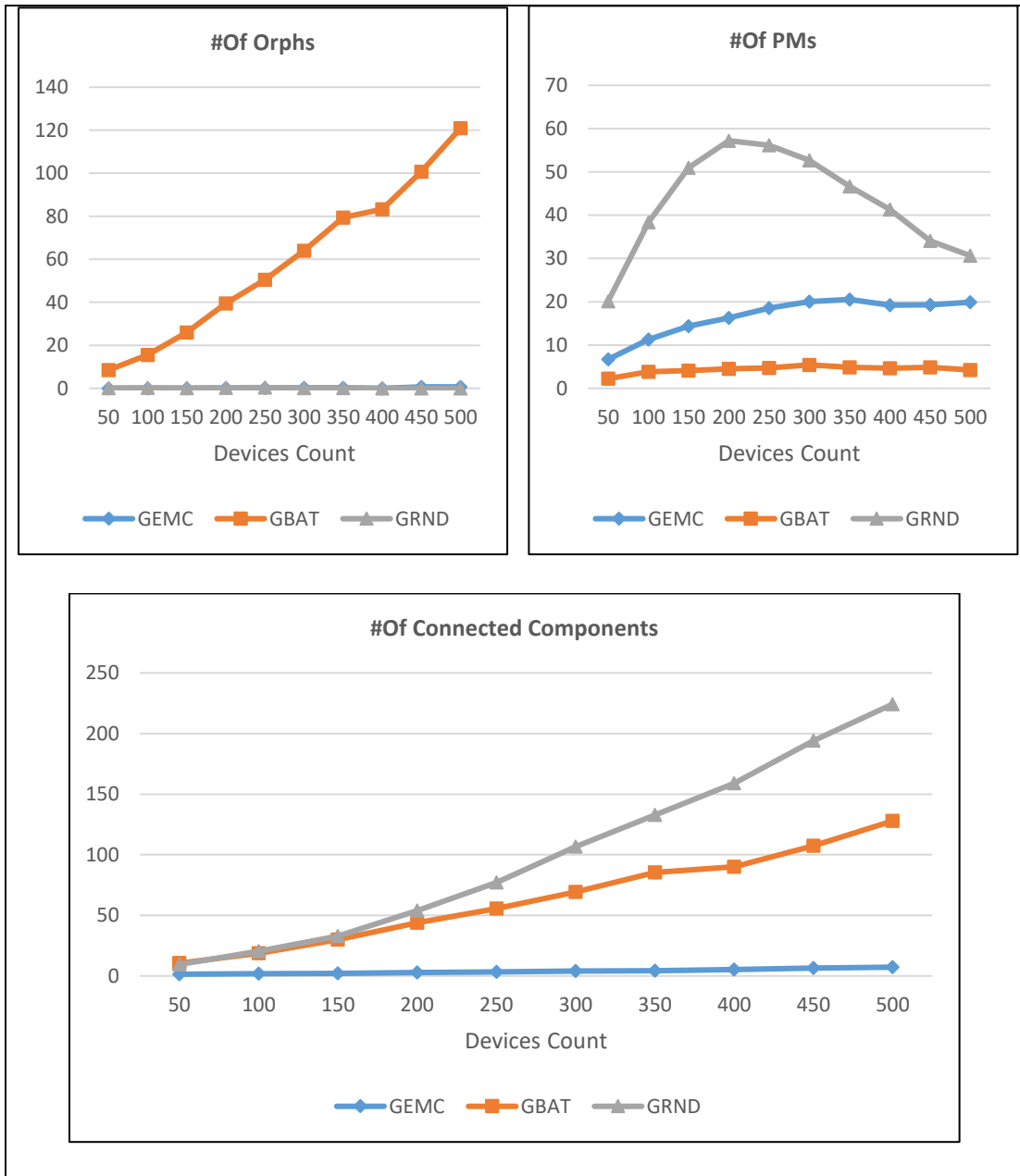


Figure 7-8 The connectivity results from GEMC, GBAT, and GRND in case of Stationary Connected Graph topology

Figure 7-9 shows that increasing the number of devices negatively affects the response time of service discovery messages, as the contention on the channel increases with the number of devices. GRND yields the worst behavior, because it produces the

largest number of GOs compared to GEMC and GBAT. Of course, the interference from that many GOs is high. GEMC and GBAT shows close and better results than GRND, as they both have lower number of GOs compared to GRND. By looking at the response of the management messages, we see that there is an increase in the response time when the density grows. GRND experiences less delays compared to GEMC and GBAT, as the boundaries for its groups are small, due to the large number of GOs. GBAT is getting the worst delays here, because it has poor decisions in defining the group boundaries. GEMC performed better than GBAT and shows ability to evenly distribute the groups.

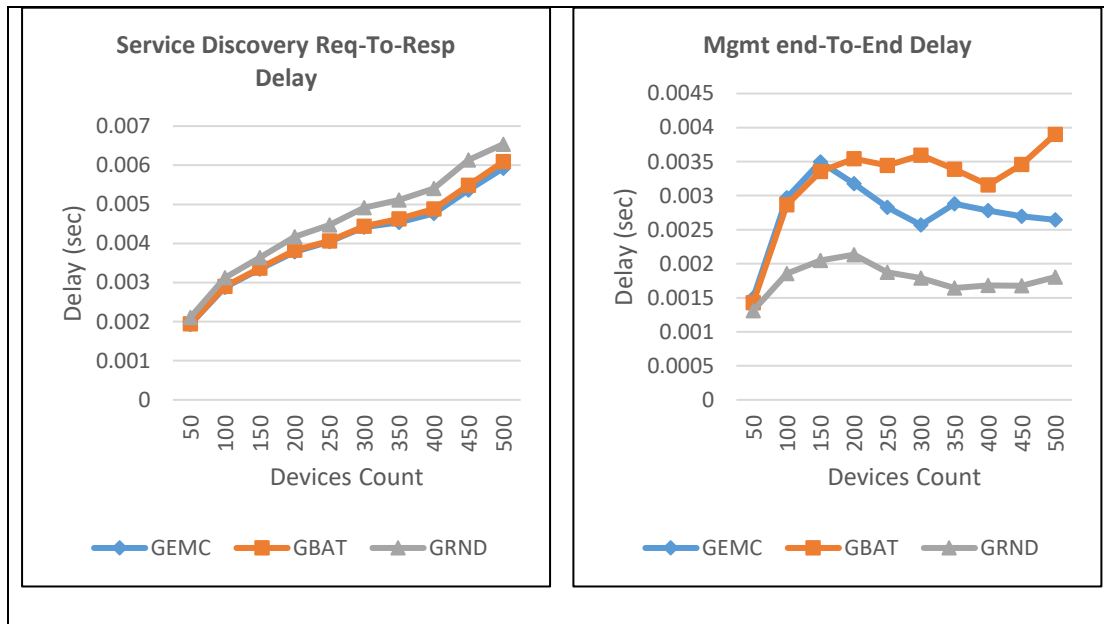


Figure 7-9 The response time results from GEMC, GBAT, and GRND in case of Stationary Connected Graph topology

Figure 7-10 shows that for all approaches, the overhead from the discovery period increases with the device density. GRND designates large number of GOs (thus,

increased number of exchanges of the SAP records), thus it shows the worst result. Compared to GRND, GEMC and GBAT are giving better results, as they are not producing too many GOs. Regarding the management messages, we see a decrease in their numbers when the density increases for all approaches, mainly due to radio interference. GEMC compared to the other two approaches can better cop with such interfaces . That is why we see increased number of management messages in GEMC.

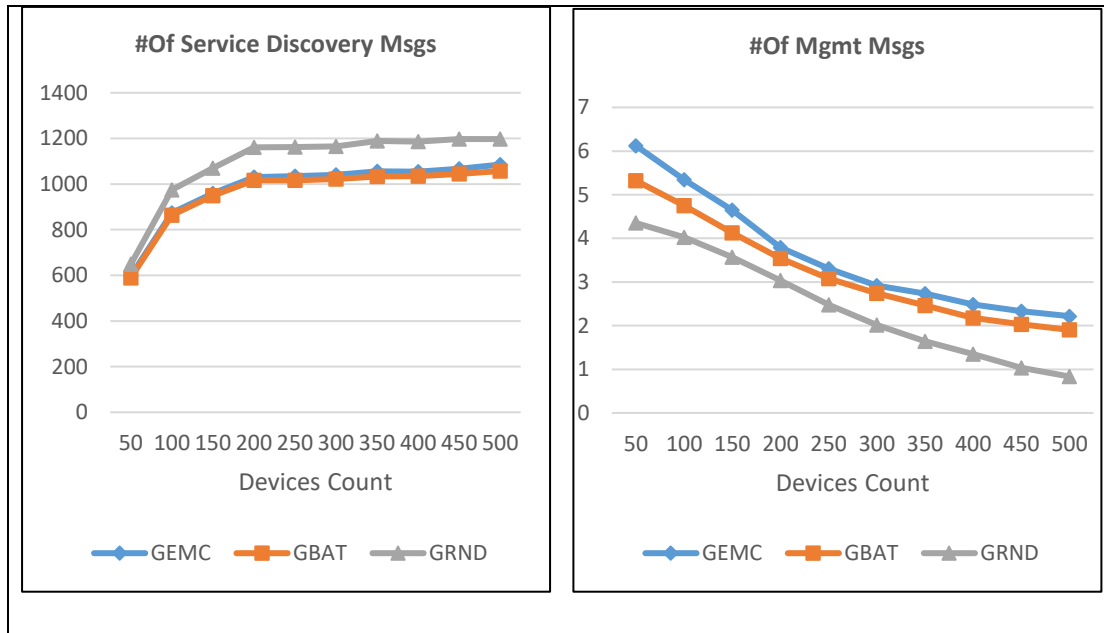


Figure 7-10 The overhead results from GEMC, GBAT, and GRND in case of Stationary Connected Graph topology

The power consumption results are shown in Figure 7-11. We notice that increasing the device density negatively affects the power consumption for the three approaches. GRND produces many GOs that cause the exchange of more messages, thus GRND leads to the highest power consumption. GBAT shows slightly better power consumption compared to GEMC since it has many orphaned nodes, which do not participate in any messages exchange. Since GEMC yields the best connectivity, as

shown in Figure 7-8, we conclude that GEMC gives the best balance between connectivity and power consumption.

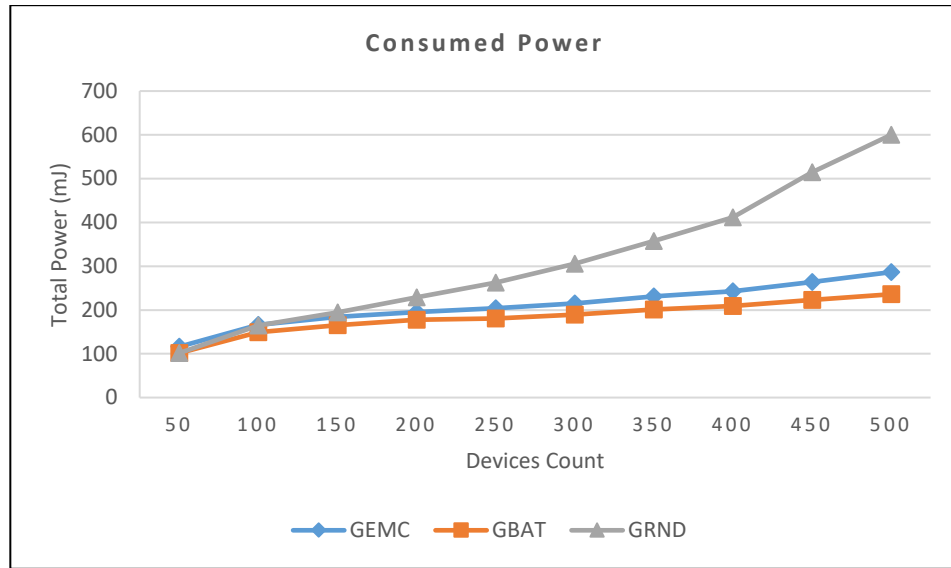


Figure 7-11 The power consumption results from GEMC, GBAT, and GRND in case of Stationary Connected Graph topology

We notice that the number of detected and resolved subnet conflicts was negligible in all cases, thus we did not find it helpful to plot it.

We conclude that GEMC is having the best balance between connectivity, message overhead, and power consumption in case of stationary connected graph topology compared to GRND and GBAT.

7.4.1.2 PM Assignment Method

In this section, we compare the PM assignment method, MUNK, of the Integrated EMC against the FRST and PRND baselines. We changed the operation of the PM selection part only and left all the default of Integrated EMC (GEMC for GO declaration, and ISNP for subnet conflict resolving). The assignment of GMs to serve

as PMs between groups is a very challenging task. If the assignment is not done correctly, the final network will partition. For this experiment, we are only interested in seeing the final number of connected components, because this the only metric that is affected by the PM selection.

7.4.1.2.1 *Static Grid*

Figure 7-12 shows the number of connected components that results from running the three different methods MUNK, FRST, and PRND. The graph shows that PRND is the worst. The reason for that is that selecting PMs at random can cause a group that is only reachable by a certain member to be disconnected due to assigning such a member to another group. For the MUNK and FRST we see that they have better results compared to PRND, but MUNK performance is slightly better than FRST. What we also notice regarding MUNK, is that the connectivity is affected by the density. For lower densities in the static grid, the number of connected components is higher than one, since there are fewer devices in the area to allow full coverage. When the device count increases to 150 or 200, we get the best coverage due to increased density. Increasing the density of devices beyond 200 negatively affects the connectivity because there are too many devices in the area which cause interference and loss of some the exchanged protocol frames; recall that the management messages count that is decreasing with density from the previous experiment. Such frame loss causes the decision of coverage to be suboptimal.

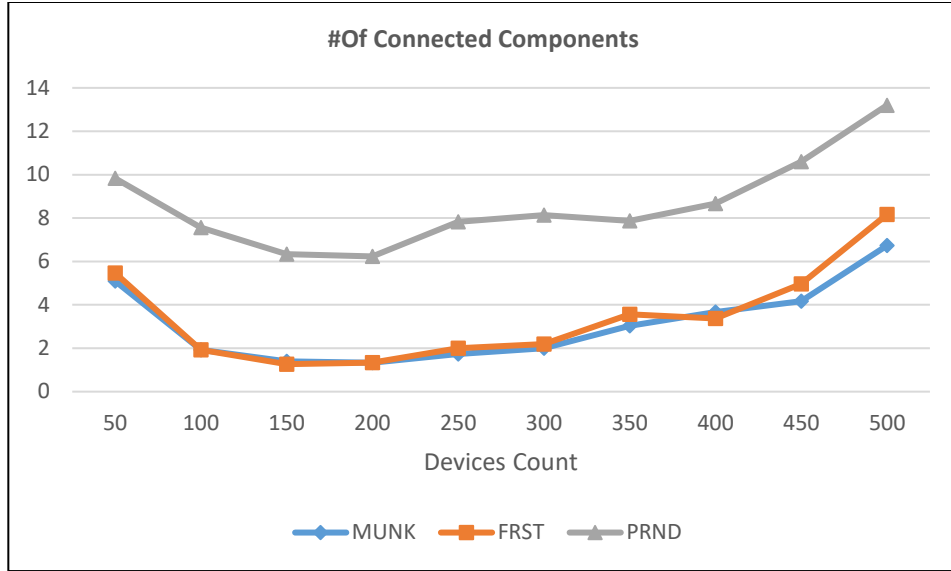


Figure 7-12 The connectivity results from MUNK, FRST, and PRND in case of Static Grid topology

From this experiment, we conclude that the density of the devices could affect the choice of PMs, due to increased interference between devices. MUNK is showing slightly better performance compared to FRST. PRND, on the other side, is giving the worst connectivity results.

7.4.1.2.2 Stationary Connected Graph

In Figure 7-13, we show the number of connected components for MUNK, FRST, and PRND. We see that the number of connected components is increasing with density for all the three methods. PRND is clearly the worst, as it randomly selects PMs. FRST and MUNK yield better results than PRND, where MUNK is slightly better. For MUNK, we get almost full connectivity with 50 devices. Increasing the number of devices negatively affects the number of connected components since in the stationary connected graph topology the devices are deployed close to each other. Due, to the increased interference that is caused by the increase in density, we start to lose certain

messages (i.e. heartbeat, and SAP records) that are required to let the GOs have a better understanding of the topology. Thus, the decisions that are taken for assigning PMs are not the optimal ones.

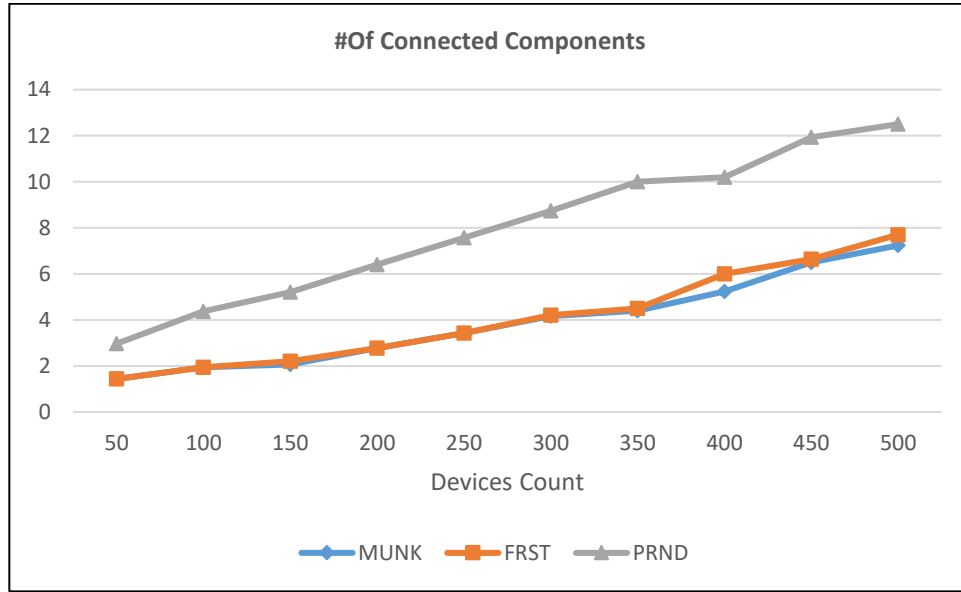


Figure 7-13 The connectivity results from MUNK, FRST, and PRND in case of Stationary Connected Graph topology

From this experiment, we conclude that the density of the devices could affect the choice of PMs, due to the increase of interference between devices. When we compare the results from this deployment type to the static grid result, we conclude that the deployment type is also affecting the connectivity. PRND is giving the worst connectivity results. MUNK is showing slightly better performance compared to FRST.

7.4.2 The Effect of Parameters

Our objective in this section is to capture the effect of changing some of the parameters on the performance of our implementation of the Integrated EMC. The

focus is on determining the values of these parameters that allow the Integrated EMC (i.e. EMC, ADS, ELN, and ISNP) to yield best performance.

7.4.2.1 Parameters Affecting Radio Links

We mainly focus on the transmission power (TxPower) and the path loss (PathLoss) since they affect connectivity and overhead. In addition, we will study the effect of mobility on our Integrated EMC implementation.

7.4.2.1.1 TxPower

In this experiment, we have set TxPower to 0.2mW and increased to 1mW in increments of 0.1mW. Lowering TxPower will enhance power consumption and radio signal interference. However, setting the TxPower below a certain threshold could have negative effect on the connectivity as shown in Figure 7-14. Basically, the number of GOs diminishes when increasing power. Small TxPower values makes the transmission range short and limits inter-device reachability. Similarly, the GM count is increasing with TxPower, which is a result of the decrement of the GO count. The number of PMs is also related to the GO count; whenever the GO count increases the PMs count increases too to have enough coverage between groups. However, the number of connected components indicates that the number of PMs was not enough to cover all groups for lower values of TxPower, especially in the connected graph topology, which explains why we start the graph with large number of disjoint components. Once the TxPower reaches 0.8-0.9mW, the connectivity improves for both topologies. The strange observation here is that once TxPower exceeds 0.9mW, the connectivity starts

to be negatively impacted again. That could be explained by the fact that having more TxPower means more interference.

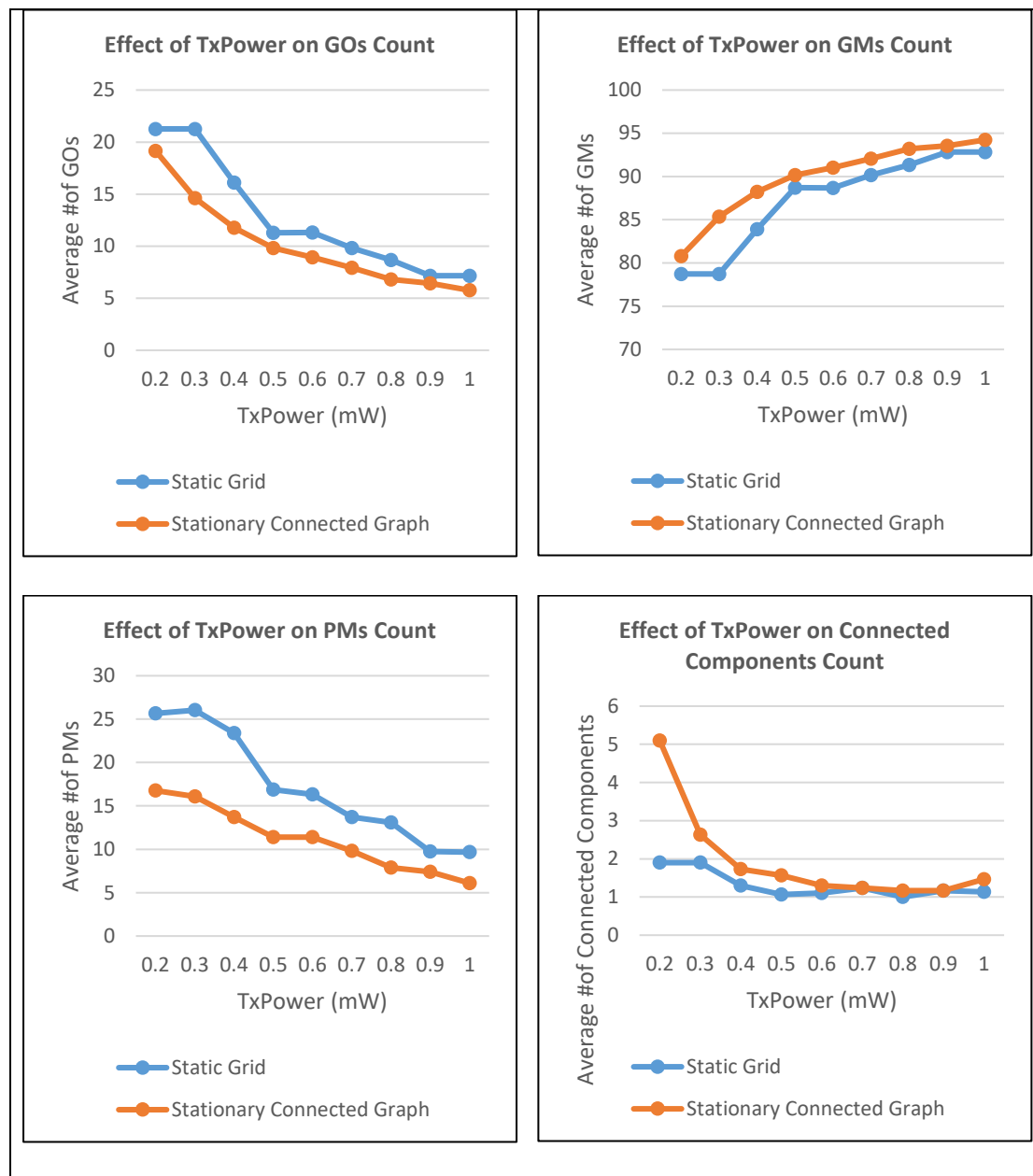


Figure 7-14 The effect of changing TxPower on connectivity

Figure 7-15 shows how TxPower affects the response time of both the service discovery part of EMC/ADS and the group management part of EMC/ELN. We notice

that the response time is negatively affected by the range of transmission. The reason for this is that increasing the power allows more devices to involve in message exchange, which cause more interference and hence affect the delays.

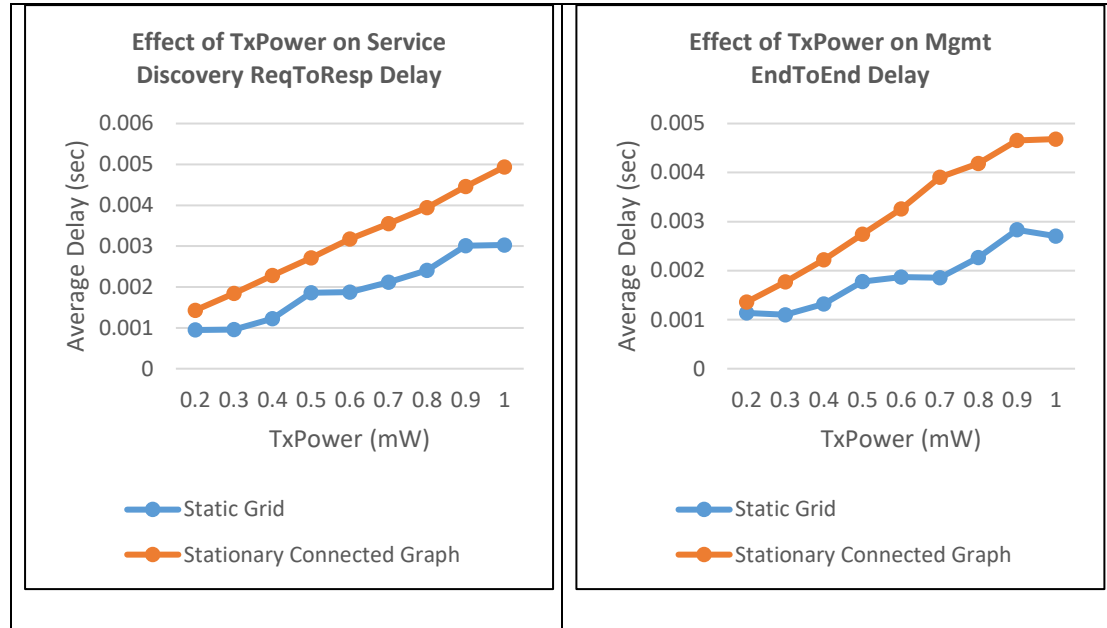


Figure 7-15 The effect of changing TxPower on response time.

Figure 7-16 shows how the TxPower affects the overhead of both the service discovery part and the group management part of EMC. What is noticed is that the number of service discovery messages is increasing proportional to TxPower. This is very much expected since having short range means less inter-device reachability and consequently less interaction between devices. However, the management messages decrease as TxPower grows. We can explain that by stating that the messages in the service discovery phase are having a broadcast nature, thus they are accepted by all reachable devices. On the other hand, the management messages are unicast messages

within the group. As the number of groups is decreasing, the number of these messages decreases as well.

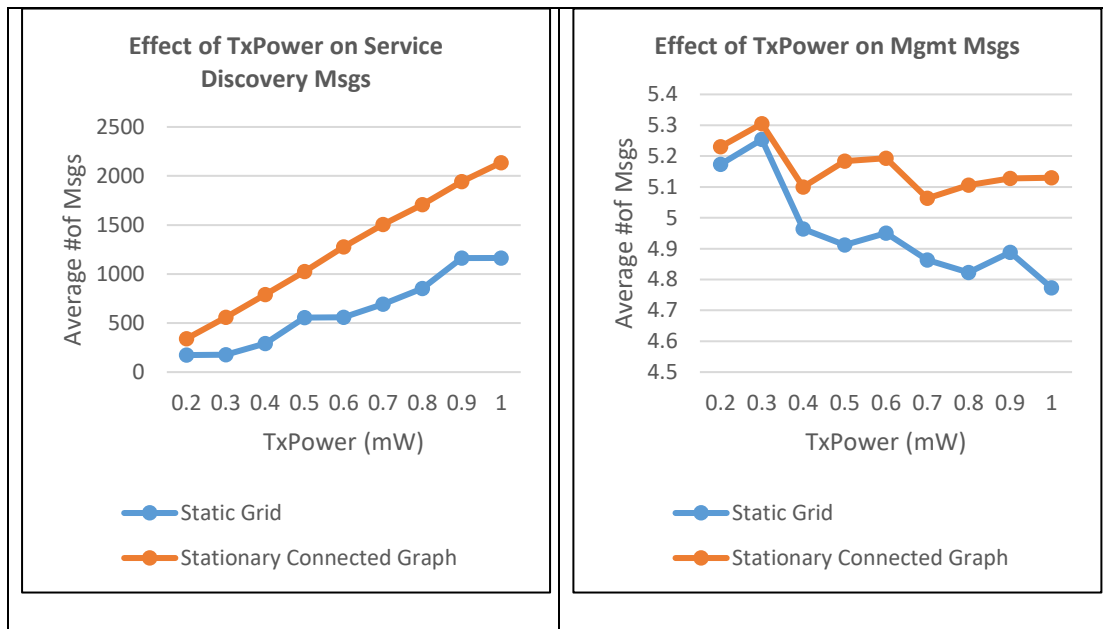


Figure 7-16 The effect of changing TxPower on Overhead

The effect of TxPower on power consumption is shown in Figure 7-17. Since the number of reachable devices increases when growing TxPower, we got more messages exchanged between devices. Such increase in the interactions between devices causes the increase in power consumption. In addition, the power that the transceiver is using to transmit is increased, which adds to the overall power consumption.

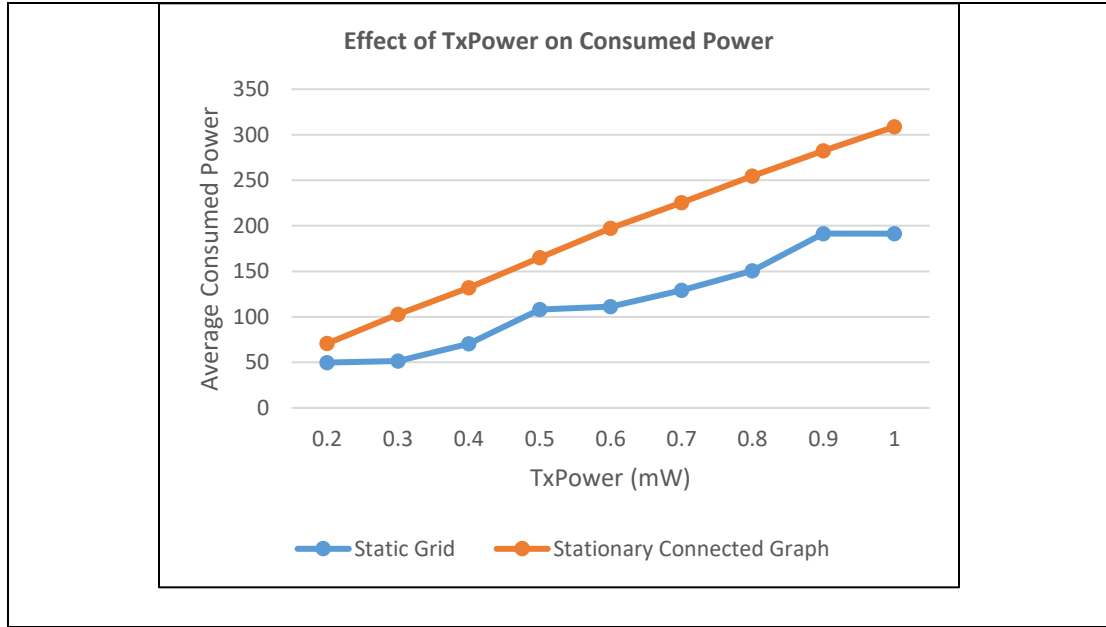
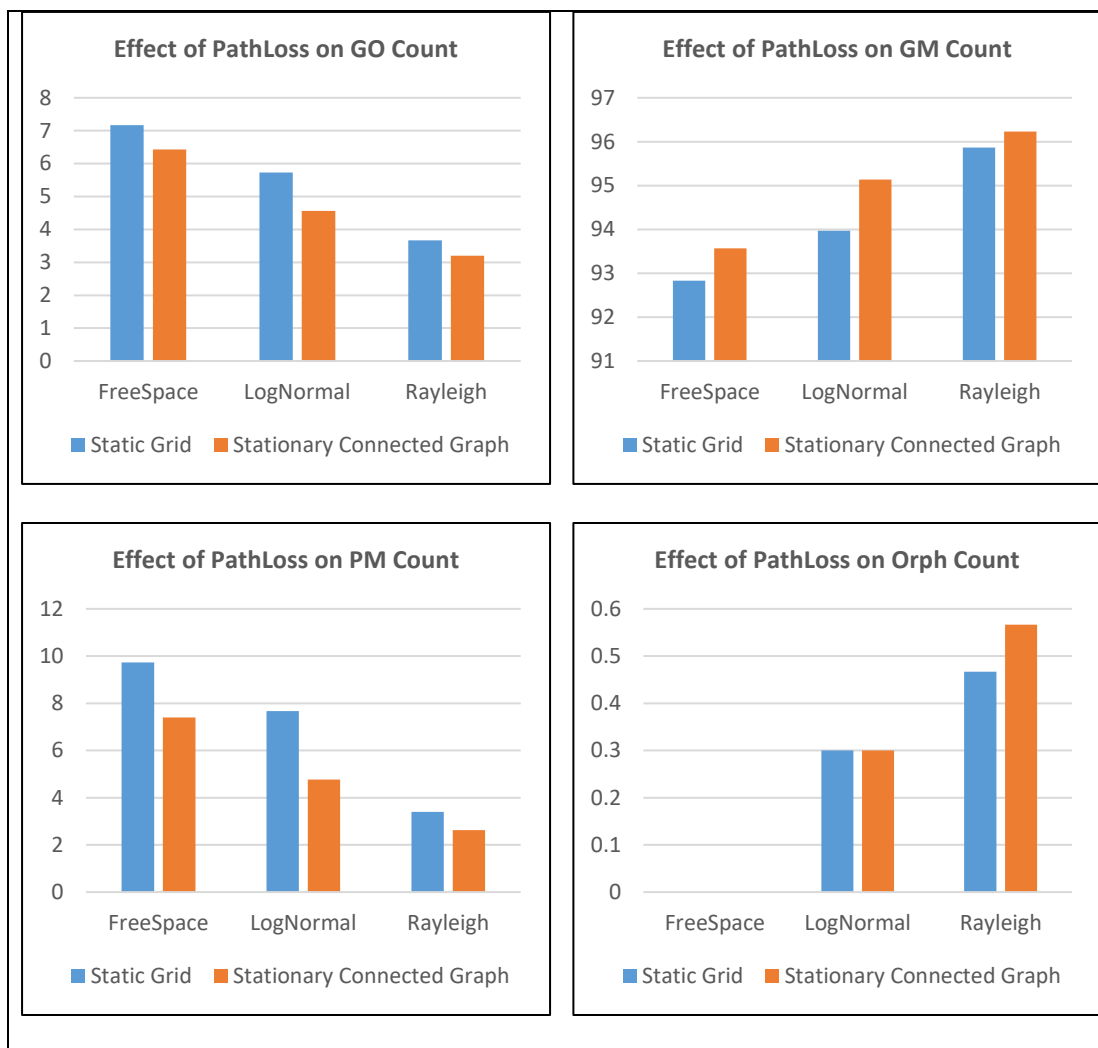


Figure 7-17 The effect of changing TxPower on power consumption

From this experiment, we could note that setting the power of transmission to 0.8 or 0.9mW would give the best results in terms of connectivity. Thus, there is a trade-off between power consumption and connectivity.

7.4.2.1.2 PathLoss

To capture the effect of the signal propagation model, we changed the *PathLoss* parameter from the FreeSpace model, the LogNormal and the Rayleigh models, which are more complex and deemed to be more practical. In Figure 7-18, we show the effect of changing the *PathLoss* on connectivity. We can notice that the complexity of the path loss model negatively affects the connectivity. The number of GOs decreases, which leads to an increment in the number of GMs. The PM count and the orphaned members have grown as well, which leads to the presence of more connected components and consequently diminished connectivity. Such an effect is more apparent under the Rayleigh model which model noisy environments.



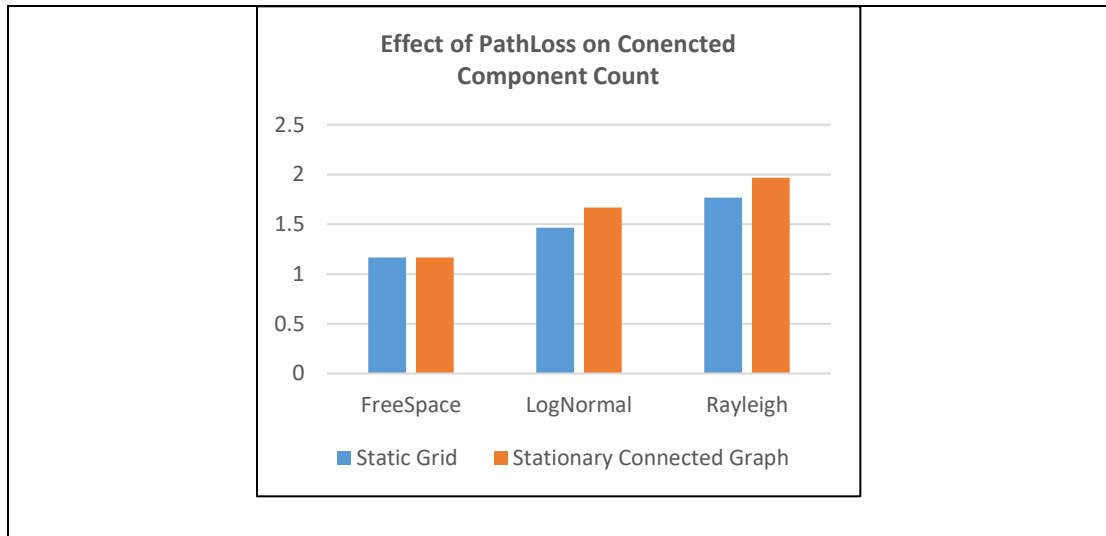


Figure 7-18 The effect of changing PathLoss on connectivity

The overhead for both service discovery and management is shown in Figure 7-19. We notice from such figure that the message overhead is decreasing as we move to PathLoss types that model noisier environment, because of having more interference.

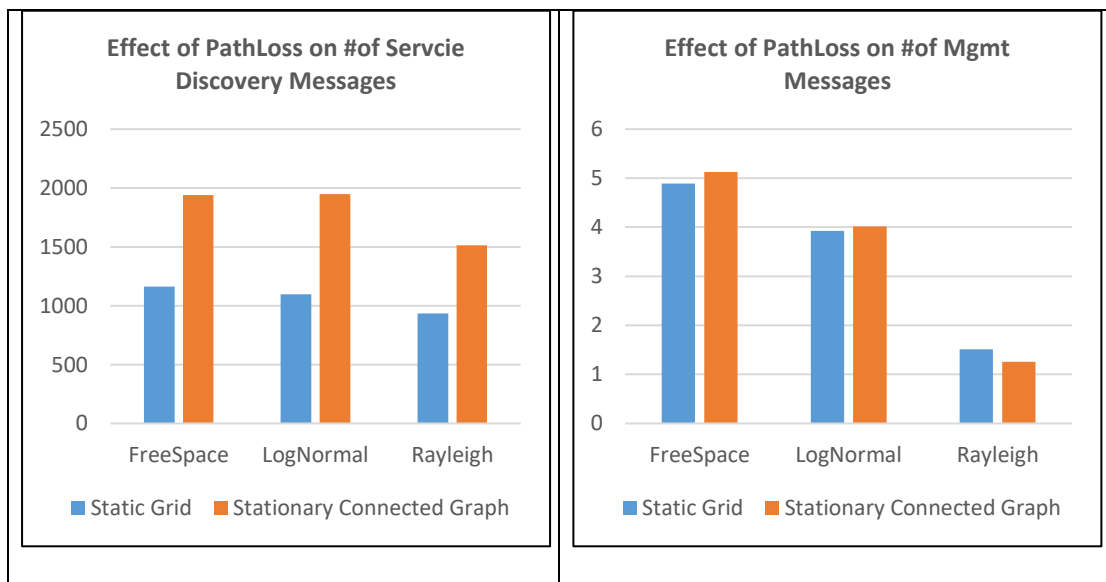


Figure 7-19 The effect of changing PathLoss on Overhead

The power consumption is decreasing with complexity as shown in Figure 7-20. This should be due to the decreased number of messages that reaches the other parties, which leads to fewer responses.

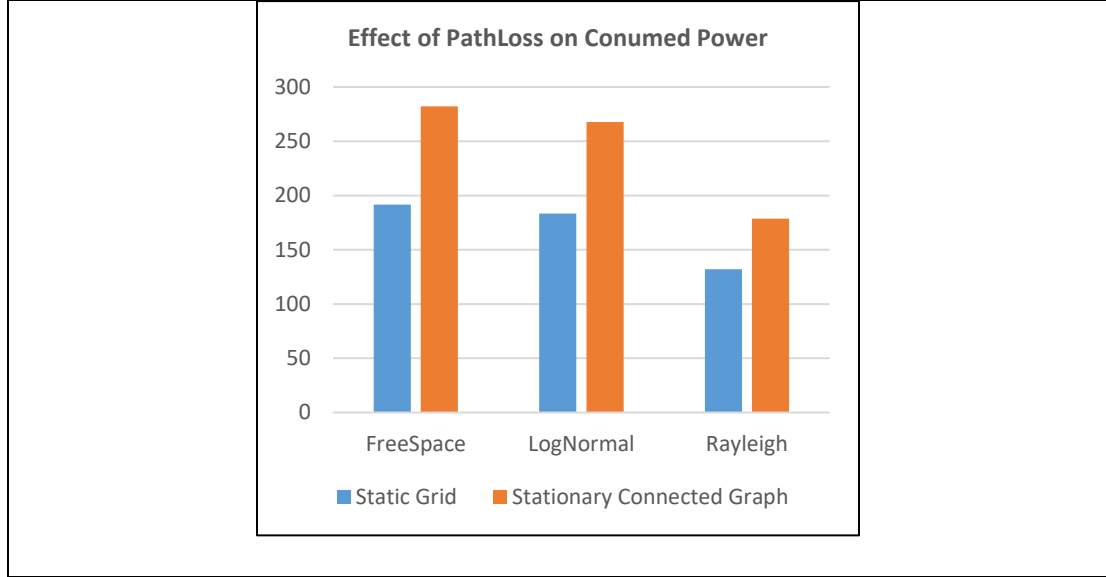


Figure 7-20 The effect of changing PathLoss on power consumption

To summarize, the path loss could negatively affect any interaction between devices when the radio signal propagation suffers increased interference.

7.4.2.1.3 Mobility

In this experiment, we capture the effect of mobility on the Integrated EMC. During this experiment, we used four types of topologies, static grid, stationary connected graph and mobility versions of them, which gives a total of four topologies. The simulation time is set to allow the Integrated EMC to run for 30 consecutive runs, i.e. $T_{simTime} = 30 \times (T_{decalreGO} + T_{selectGO} + T_{pxAssignment} + T_{teardown})$. As in all other experiments, we repeated such experiment 30 times with different seed to get the average. The mobility model that we applied is a model that captures the movement of 100 people

in an area of $500\text{m} \times 500\text{m}$, a space that can be envisioned as a public park or a museum. The parameter that we set for such person mobility model are shown in Table 7-3.

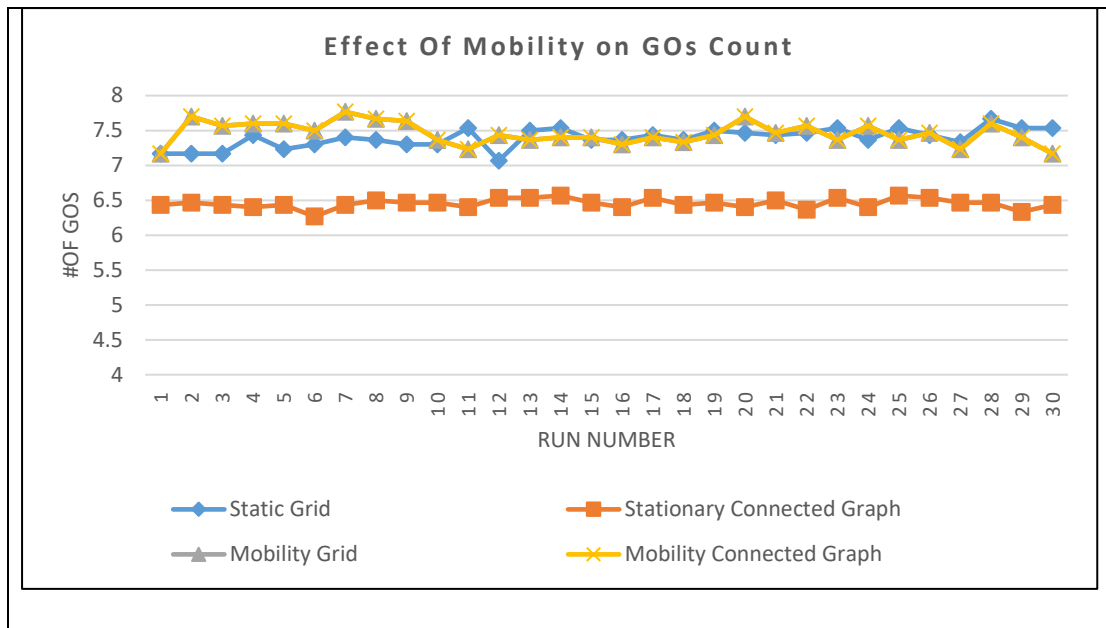
Because of the high power-consumption of the GOs, new devices could have better ranks and serve as GOs in the next round. With this experiment, we also capture the GO role rotation in both the static and mobile scenarios, which happens when we restart the protocol. Such role rotation could lead to different connectivity results, especially in the mobility case. The results of this experiment are shown below.

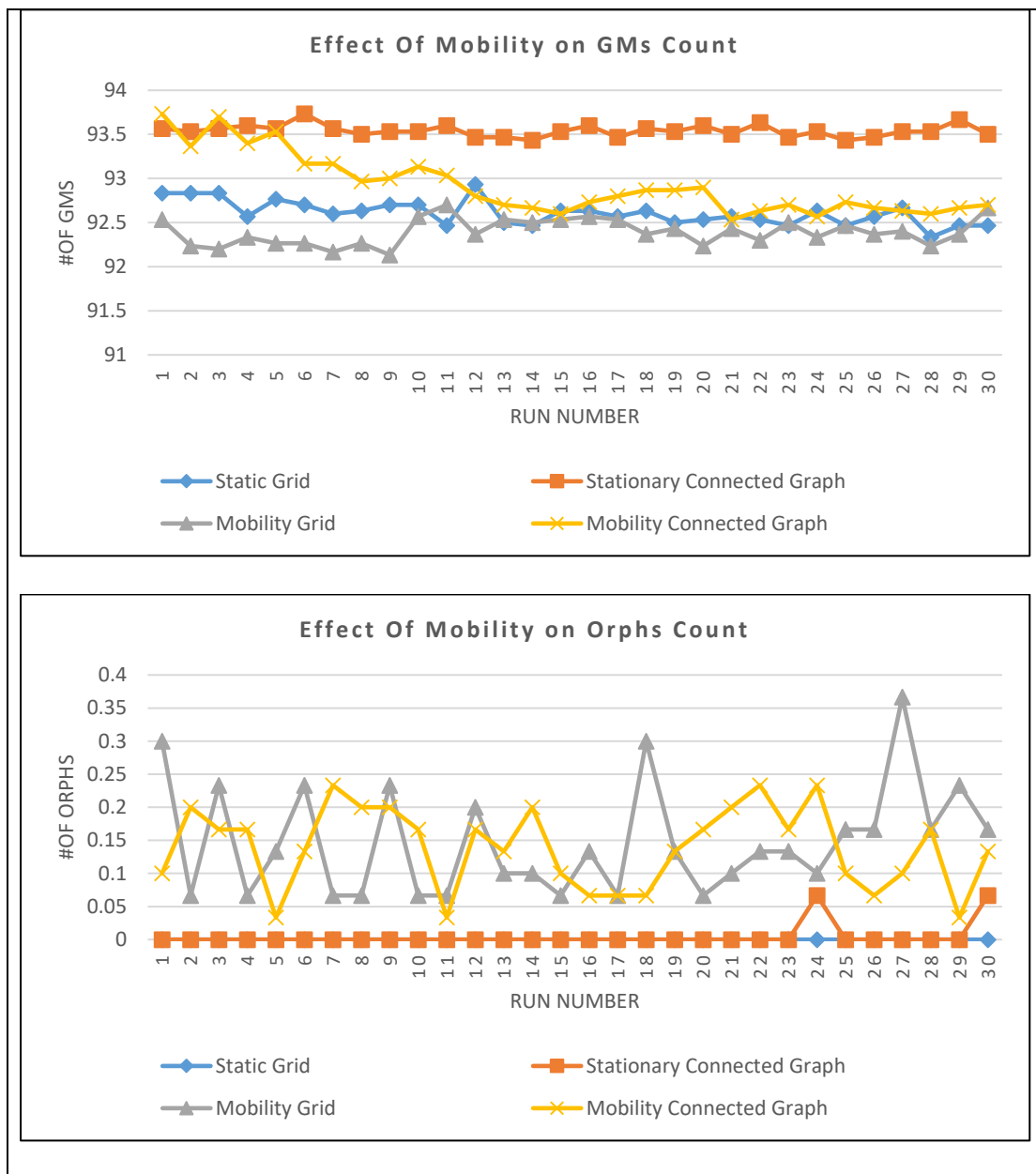
Table 7-3 The mobility parameters used to model person movement

Mobility Parameter	Change of Value
Speed	Exponential distribution of 1.3 meters/sec
Angle	Zero with a variance of 5 degrees
Change Interval	2s with a variance of 0.25s

Figure 7-21 shows the effect of mobility and GO role rotation on coverage. We notice that the number of GOs and GMs is not changing widely in the case of stationary connected graph. This is because more devices are in the range of each other in this topology, thus the decisions stay nearly the same. However, in the static grid and the mobility cases, we see that there is a noticeable change in the number of GOs and PMs. The edge devices in the static grid case could be the reason for such a change, as they may have better ranks. These edge devices do not cover as much areas as other devices, which lead to gaps in coverage that requires more GOs to cover. For the mobility cases, varying the device location affects the covered area, thus the GO and GM counts are

changing to adapt with that. We notice also that the number of orphaned devices is affected also by mobility, due to the possibility of having devices move before completing their role negotiation successfully. The PM count is changing with each round of the protocol for all topologies. This is because changing the GOs affects the group boundaries, thus changing the overlapping region between groups, which contains the devices that can link the groups. Because of the change in PM count, the number of connected components changes, as seen from the figure. However, we notice that such a change is worse in case of mobility, which is expected.





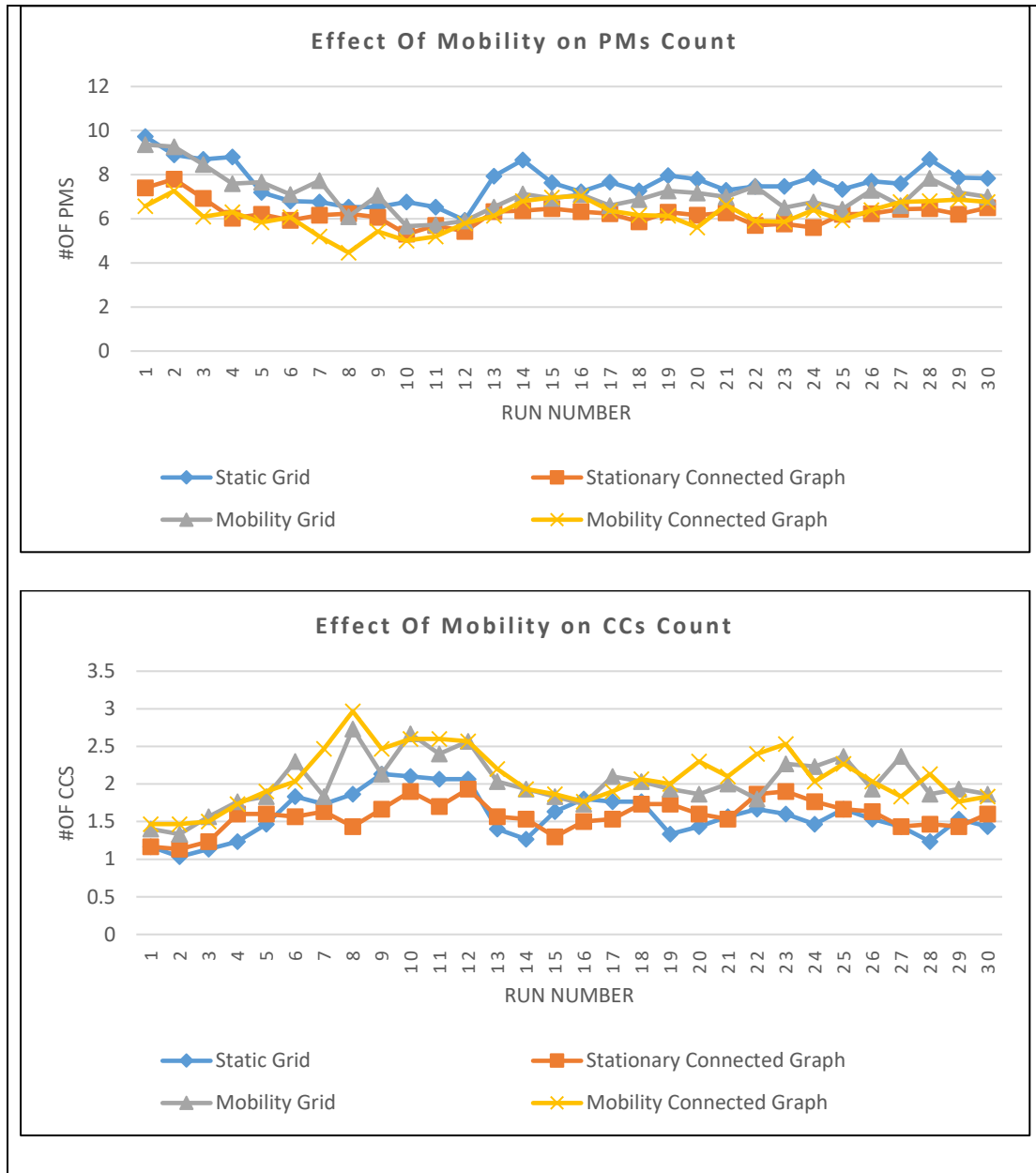


Figure 7-21 The effect of Mobility on connectivity

Figure 7-22 and Figure 7-23 show the effect of mobility on response time and overhead. For the service discovery, the response time is enhanced in case of mobility. Such decrease in delay could be due to the change in the device locations that leads to spreading of the devices in a way that could boost the response time. The same applies

to the number of service discovery messages. When considering the management delay, we notice that mobility version of the static grid has better responses, which could be correlated with the decrease in the number of GMs in case of the grid. The same effect is seen for the management messages in both mentioned cases. Regarding the stationary connected graph and its mobile version, we notice that the mobility negatively affects the management end-to-end delay. However, the number of management messages are decreasing in case of mobility, because of having a fewer number of GMs, which leads to fewer interactions within the groups.

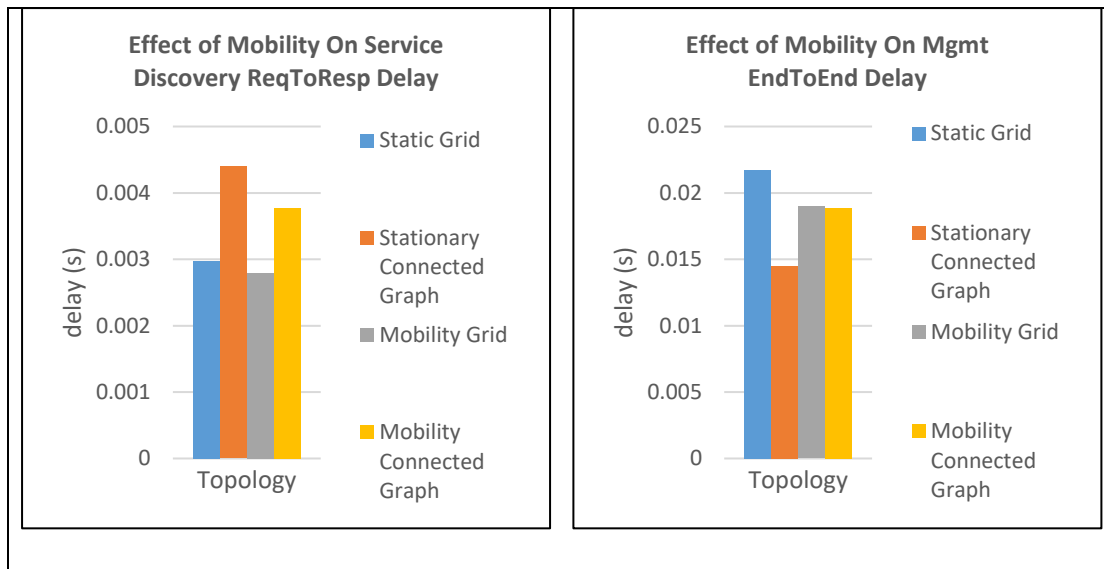


Figure 7-22 The effect of Mobility on response time

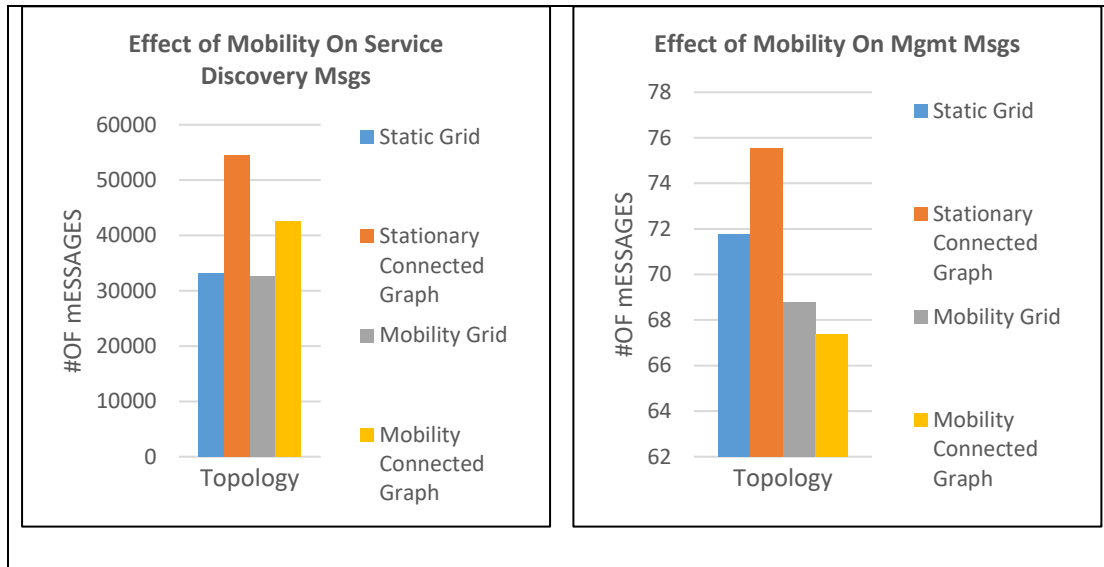


Figure 7-23 The effect of Mobility on overhead

Figure 7-24 shows the effect of mobility on power consumption. What we see here is that the pattern of power consumption is following the pattern of service discovery messages which is dominating. We are not considering here the power consumed due to moving, as we assume that the devices are carried by persons not mobile by themselves.

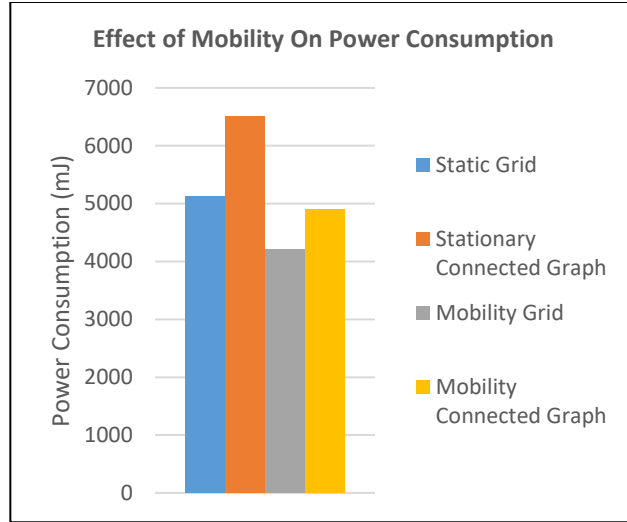


Figure 7-24 The effect of Mobility on power consumption

At last, the mobility and the changing of the GO role between devices create certain challenges that could cause changes in coverage. The power consumption and the message overhead has enhanced due to the redistribution of devices. However, the delays show some mixed results. As a result of this experiment, we plan to extend our work in the future to better accommodate the effects of mobility and role changing of GOs.

7.4.2.2 Protocol Parameters

Certain parameters of our protocols affect the various performance metrics as summarized in Table 7-4. We will show the simulation results for each parameter in the balance of this section.

Table 7-4 The effect of paramters on Performance

Parameter	Protocol	Performance Metric
-----------	----------	--------------------

		Connectivity	Response Time	Overhead	Power Consumption	Subnet Conflicts
$T_{sendInterval}$	ADS	×		xx	xx	×
$T_{declareGO}$	EMC	×		×	×	xx
$T_{selectGO}$	EMC	×		×	×	
$T_{HeartBeatGM}$	ELN	×		xx	×	
$T_{HeartBeatGO}$	ELN	×		×	×	
$T_{pxAssignment}$	EMC	xx				
$MaxX$, $MaxY$	ISNP					xx

7.4.2.2.1 $T_{sendInterval}$

In this experiment, we are changing the interval at which devices ask for service discovery records by 0.1s steps starting from 0.1s and going up to 1s. Reducing this interval while at the same time fixing the period that the devices spend doing service discovery operations means that more requests and more responses are going to be generated. Figure 7-25, shows that the effect of $T_{sendInterval}$ on connectivity is minimal, where the number of GOs, GMs and PMs stay almost the same. We found that there are no orphaned nodes; thus, we did not plot it. Also, the number of connected components is nearly the same. This result is expected, as changing this value directly affecting the number of service discovery request. As the service discovery period is fixed at 4 secs in this experiment, we will have at least 4 requests in the worst case, which is more

than enough to get complete information from neighbors. For the best case at $T_{sendInterval}$ equals 0.1 sec, we get 40 requests, which too much.

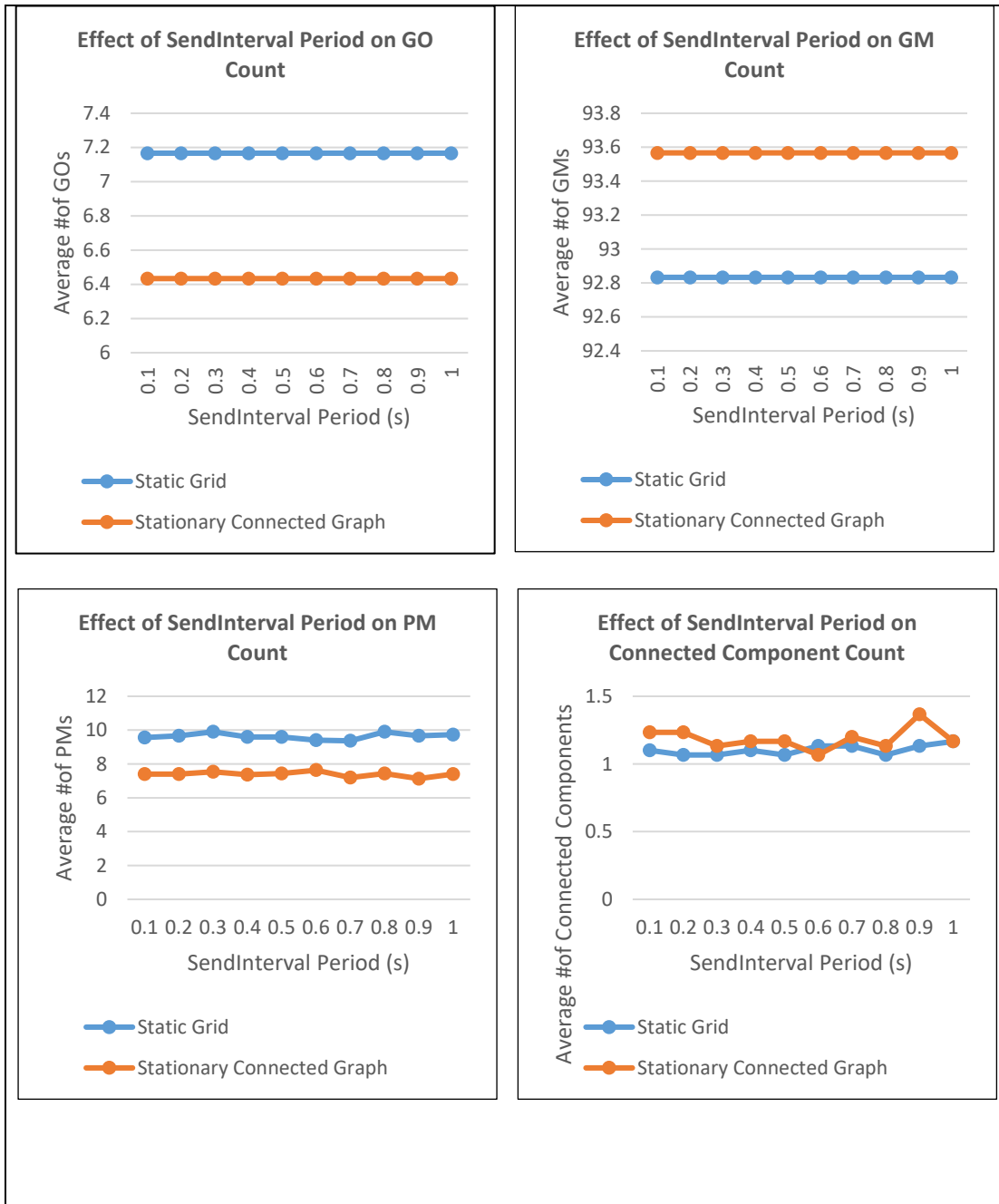


Figure 7-25 The effect of changing $T_{sendInterval}$ on connectivity

On Figure 7-26, we show the effect of changing the send interval on the service discovery overhead; the management overhead is not relevant in this case. We see that the overhead is decreasing when we increase this interval, which is a result of having fewer requests. In the far-left side of the graph, when the send interval is set to 0.1s, we get the worst message overhead, as there are too many requests generated at such interval.

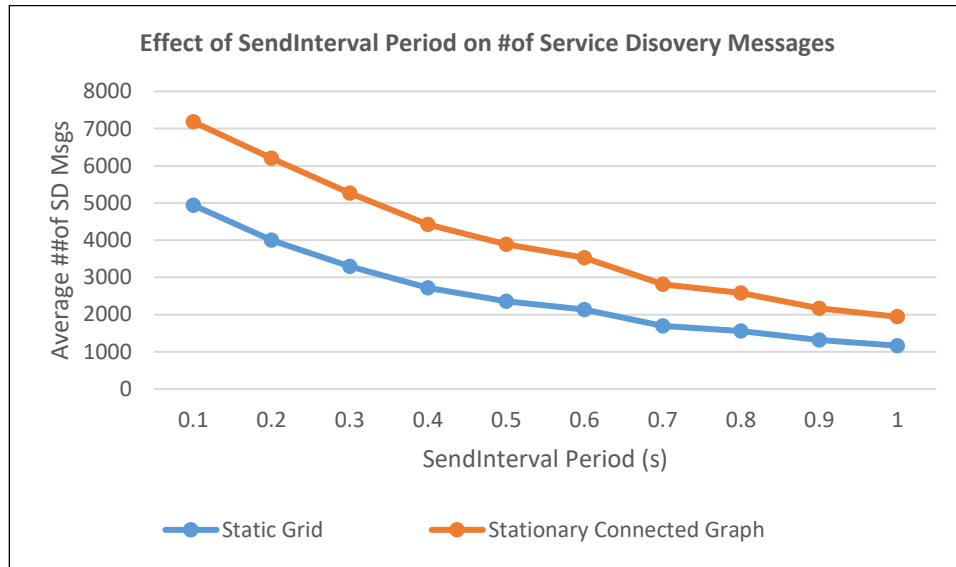


Figure 7-26 The effect of changing $T_{sendInterval}$ on overhead

The effect on power consumption is shown in Figure 7-27. The larger the send interval is the less the power consumption becomes. Of course, having fewer requests means fewer responses and less power. On the other hand, decreasing such interval increases the number of requests, which negatively affects the power consumption.

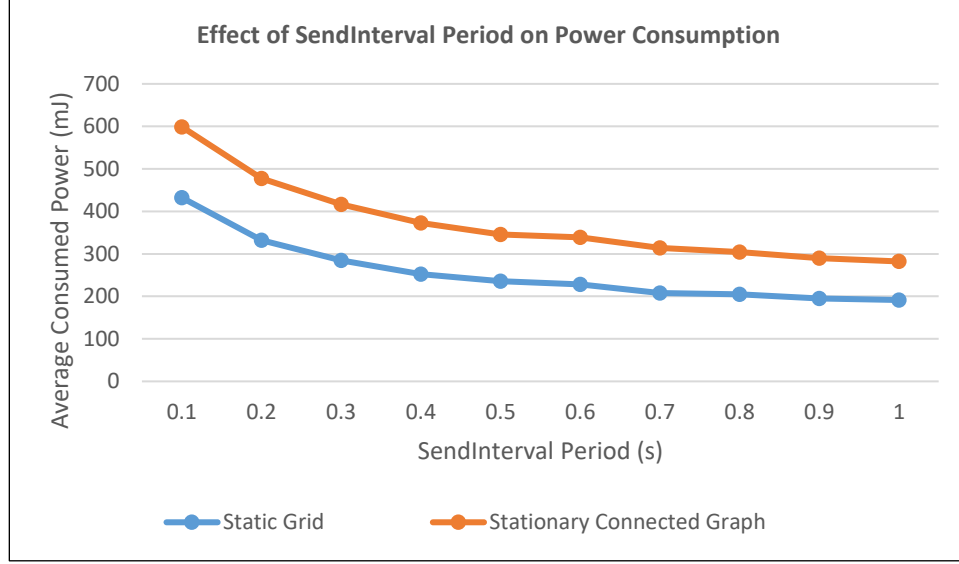


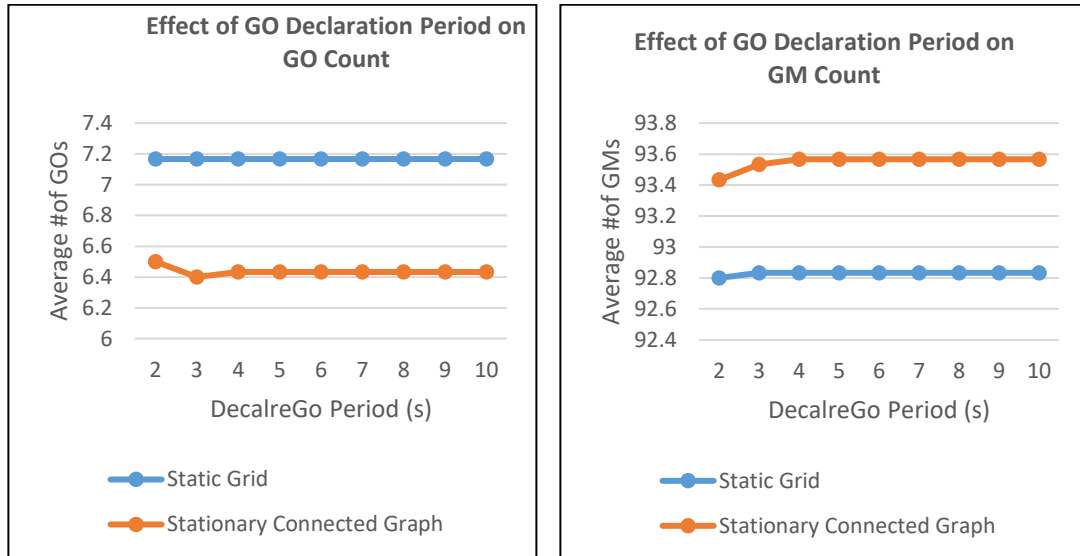
Figure 7-27 The effect of changing $T_{sendInterval}$ on power consumption

From this experiment, we conclude that $T_{sendInterval}$ should be set around the value that leads to sufficient number of service discovery requests would yield the best connectivity, overhead, power consumption. However, setting it too low has a bad impact on overhead, and power consumption. The connectivity does not change with the change of this parameter.

7.4.2.2.2 $T_{declareGO}$

To study the effect of $T_{declareGO}$, we set it to 2s then changed it up to 10s in 1s increments. This period defines the time allowed for devices to decide who will take the GO responsibility. As the $T_{sendInterval}$ is 1s, the lowest value of $T_{declareGO}$ gives only 2 possible service discovery data exchanges between devices, while for the highest value of $T_{declareGO}$ we get 10 possible exchanges. We see from Figure 7-28 that increasing $T_{declareGO}$ value positively affects the connectivity. The number of GOs is stabilized after having a value of 4s, which means 4 service discovery exchanges are

sufficient for getting a stable GO count that can sufficiently cover the area. The same applies for GM count, which stabilizes at 4s. The PM count is increasing when $T_{declareGO}$ grows and nearly stabilizes at 4s; however, it swings a little around 8s for the stationary connected graph topology. Of course, at 4s, the devices get sufficient data exchanges to negotiate their roles; however, increasing that interval beyond 4s will give redundant data exchanges, thus we see the stabilization on the curves. The number of PMs directly affects the number of connected components, so we notice that when the PM count swings at 8s the connected components are affected. For the static grid, the number of connected components approached one and did not change after the value of 4s. we notice also that for values lower than 4s, there are some orphaned members, which is a result of not having enough data exchanges at the GO Declaration period. Thus, the proposed GO filed that is supposed to fix the orphaned members case did not have the time to propagate.



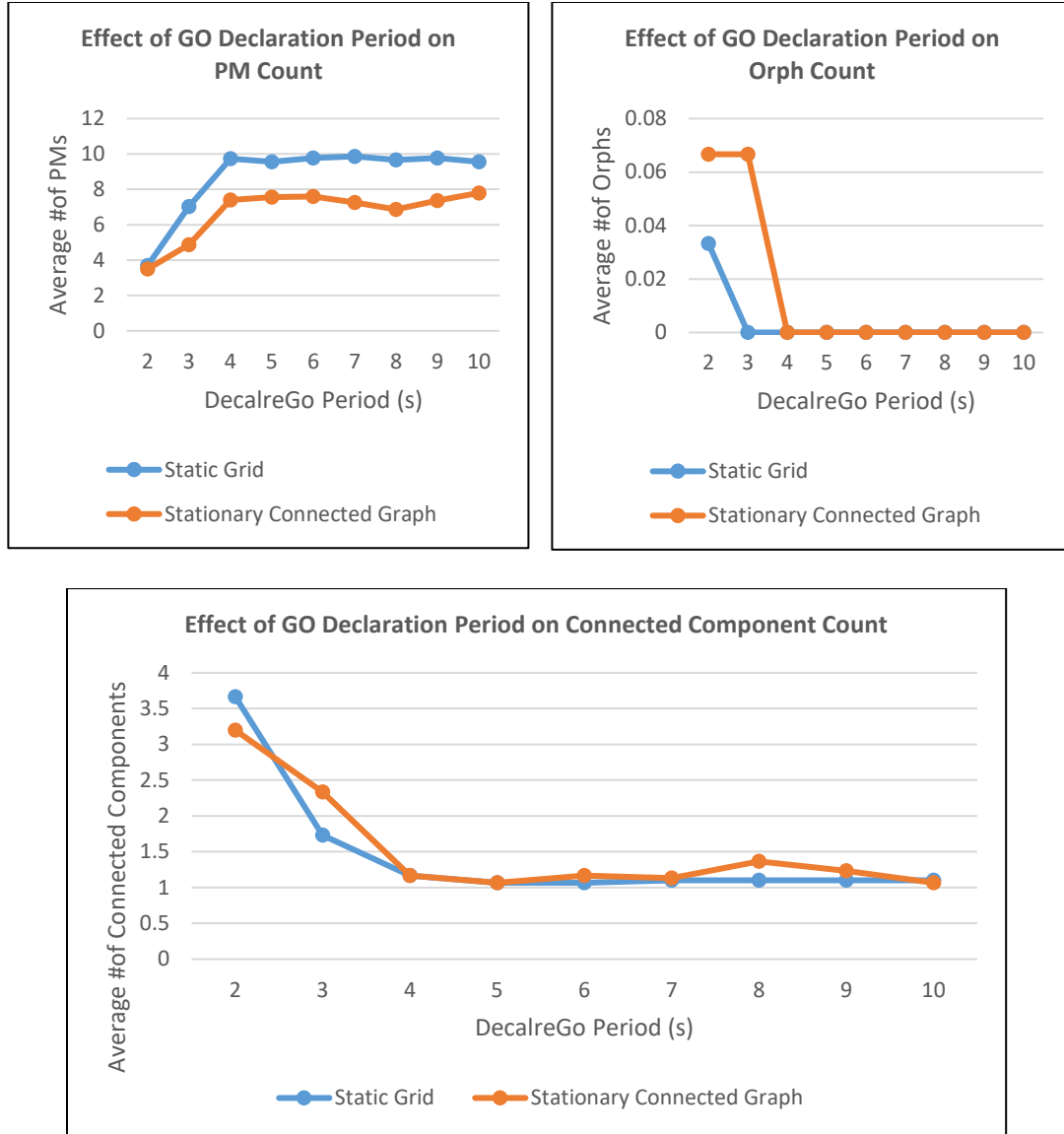


Figure 7-28 The effect of changing $T_{declareGO}$ on connectivity

Regarding the overhead, we see in Figure 7-29 the number of service discovery messages is increasing with the increase in $T_{declareGO}$; this is very much expected, as there are more service discovery exchanges. From the figure, also, we notice that the management messages overhead starts with a small value when the connectivity was

not high, afterwards it is increased and stabilized. The reason for such behavior is that, below 4s, there are some orphaned devices and the GM count is lower than other intervals, which means that fewer management interactions.

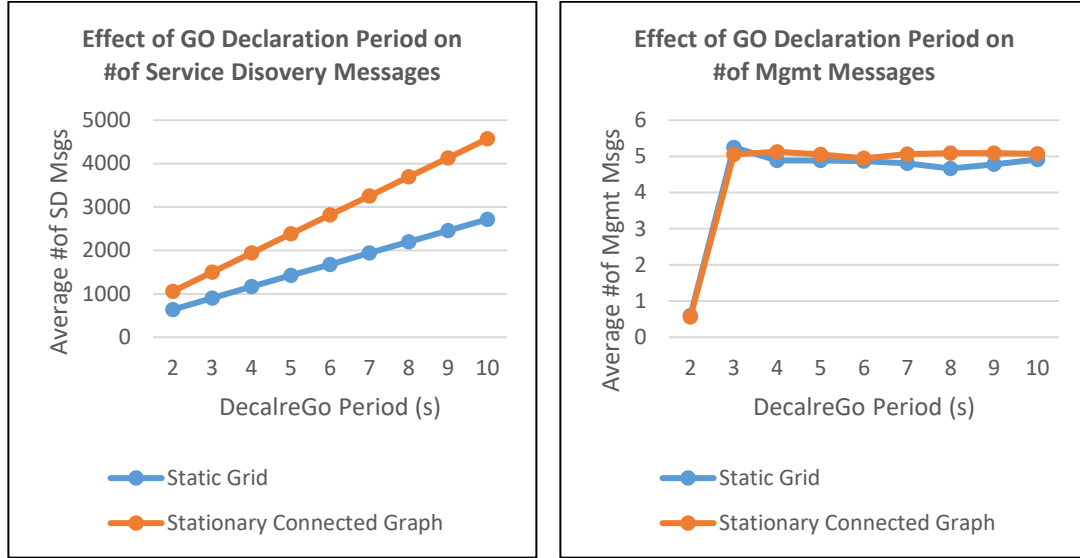


Figure 7-29 The effect of changing $T_{declareGO}$ on overhead

The power consumption, as shown in Figure 7-30, is increasing because of the increase of message overhead shown in the previous figure. We can see that the consumption starts with a low value then a big jump happens after 3s following the message overhead and connectivity patterns. As the connectivity, does not change much after 4s, we recommend setting the $T_{declareGO}$ interval to a value that gives just enough service discovery exchanges to enhance the power consumption.

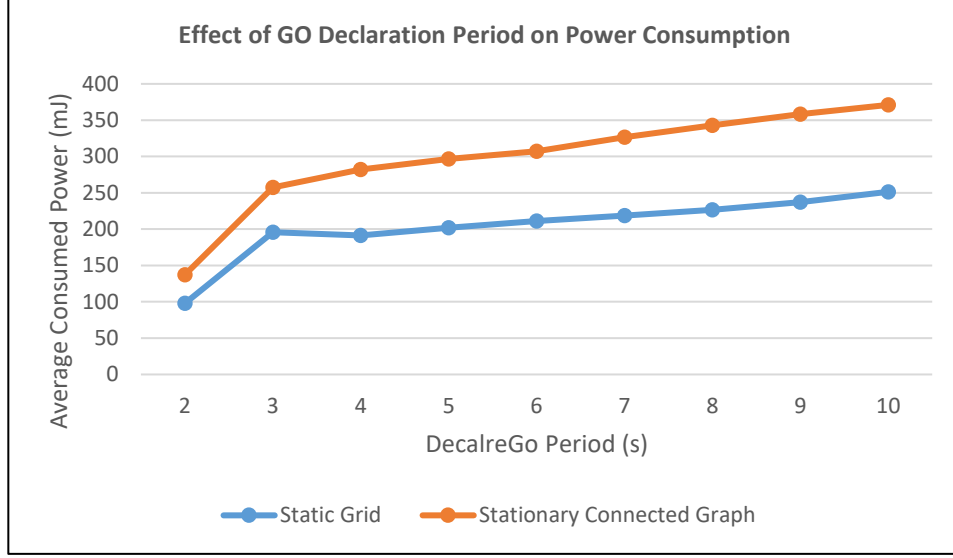


Figure 7-30 The effect of changing $T_{declareGO}$ on power consumption

The conclusion from this experiment is that giving enough time for the devices in the GO declaration period yields better results in terms of connectivity. To control the power consumption and overhead, we need not to set this parameter too high. A value of 4s seems sufficient in this experiment.

7.4.2.2.3 $T_{selectGO}$

This parameter defines the length of the period that the GMs take to select their groups. This time should be large enough to allow the reception of the SAP records from all reachable GOs. However, extending this period is not desirable in order to expedite the start of intra-group interactions. We could notice from Figure 7-31 that the PM and connected component count are changing only slightly with the change of the $T_{selectGO}$ parameter.

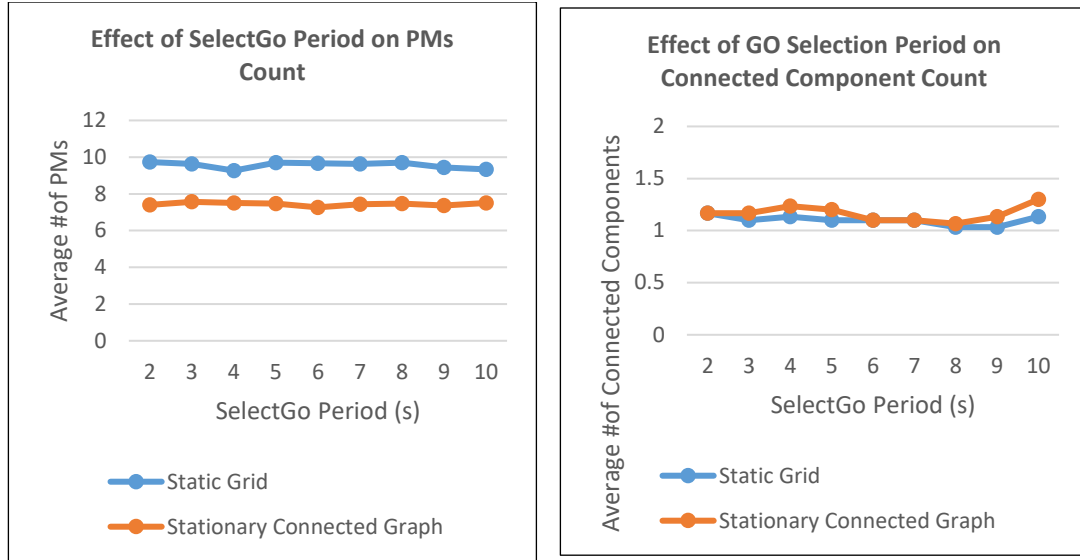


Figure 7-31 The effect of changing $T_{selectGO}$ on connectivity

What we see from Figure 7-32 and Figure 7-33 is that the overhead of the discovery service period and the power consumption are increasing when $T_{selectGO}$ is increased. This result is expected, as the devices exchange more service discovery records when we extend this period, and consequently more power is consumed.

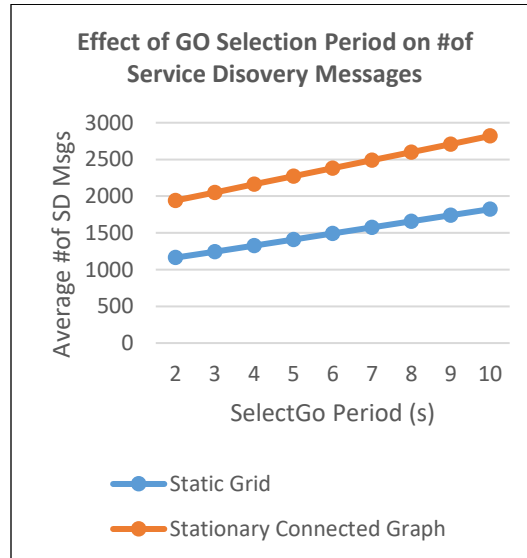


Figure 7-32 The effect of changing $T_{selectGO}$ on overhead

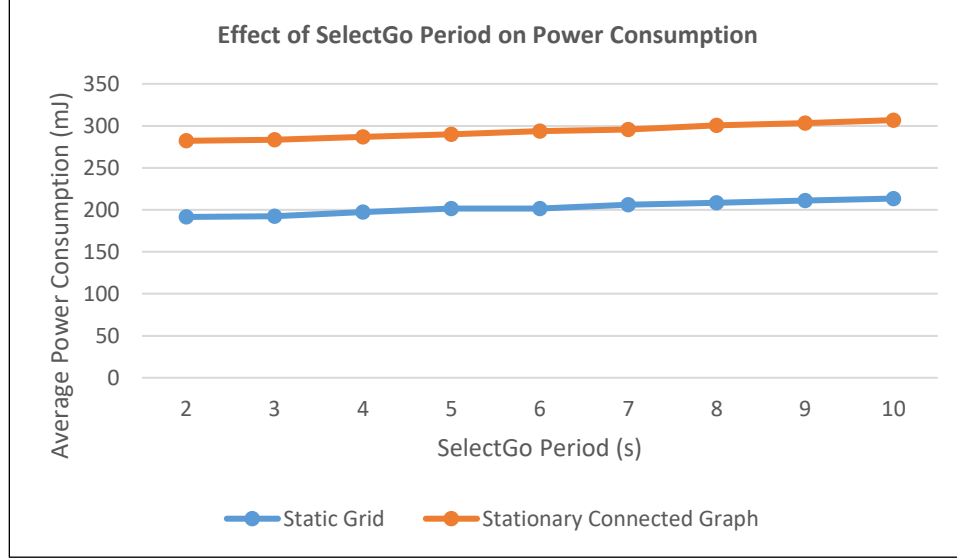


Figure 7-33 The effect of changing $T_{selectGO}$ on power consumption

From this experiment, we conclude that when choosing the $T_{selectGO}$ value we should minimize its length to reduce overhead and power consumption. There is no gain from increasing the length of this period in terms of connectivity. Based on this experiment, setting $T_{selectGO}$ to 2s seems to be suitable.

7.4.2.2.4 $T_{HeartBeatGM}$

. During this experiment, we increased $T_{HeartBeatGM}$ from 0.1s to 1s with 0.1s in between. Decreasing such a value, while keeping the $T_{HeartBeatGO}$ period the same increases the number of message from GM to GO. Figure 7-34 shows the effect of such a parameter on connectivity. We notice that at the beginning, where the number of heartbeat messages are the highest, the number of PMs slightly grows. Increasing $T_{HeartBeatGM}$ reduces the PM count; nonetheless, it is almost stable around 10 and 8 for static grid and stationary connected graph topologies, respectively. The connected components number is fluctuating around a value of 1.1 most of the time. It is slightly worse at

larger values of $T_{HeartBeatGM}$. As there is not much of change in connectivity, we see that we do not need to set this parameter too low to reduce the number of management messages. This would enhance the power consumption, as we will see next.

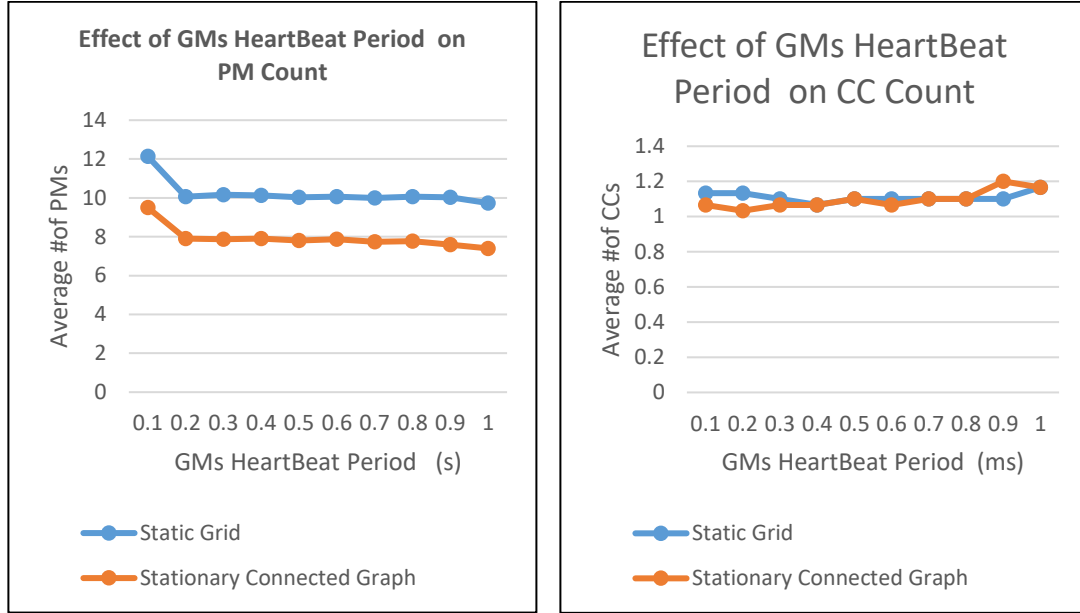


Figure 7-34 The effect of changing $T_{HeartBeatGM}$ on connectivity

From Figure 7-35, we see that the management overhead is diminishing with larger values of $T_{HeartBeatGM}$; the service discovery overhead is not relevant here. It is desirable to reduce such overhead, so it seems that setting $T_{HeartBeatGM}$ to 1s gives the best result.

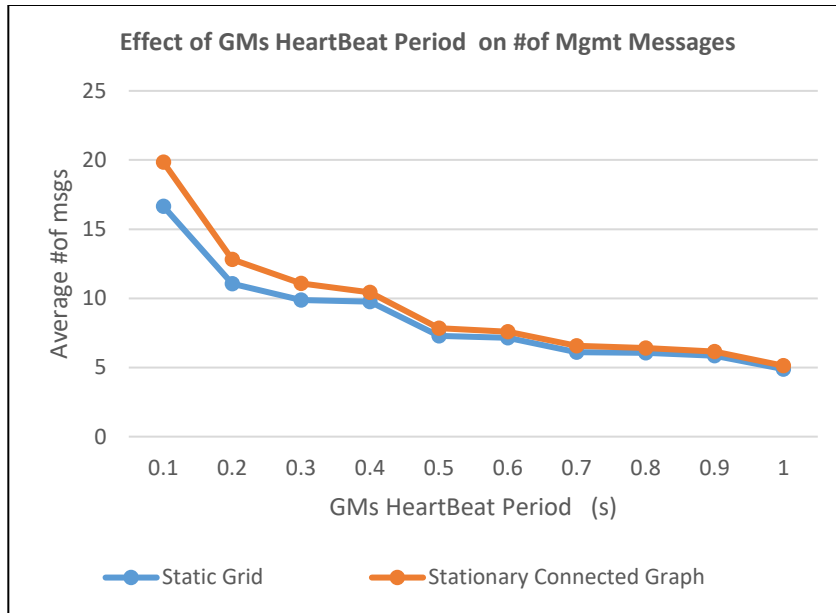


Figure 7-35 The effect of changing $T_{HeartBeatGM}$ on overhead

The response time is not a metric that would be impacted by this parameter, thus we did not study it. In Figure 7-36, we find that the power consumption is enhanced when we increase the value of $T_{HeartBeatGM}$. At 1s we get the lowest power consumption value.

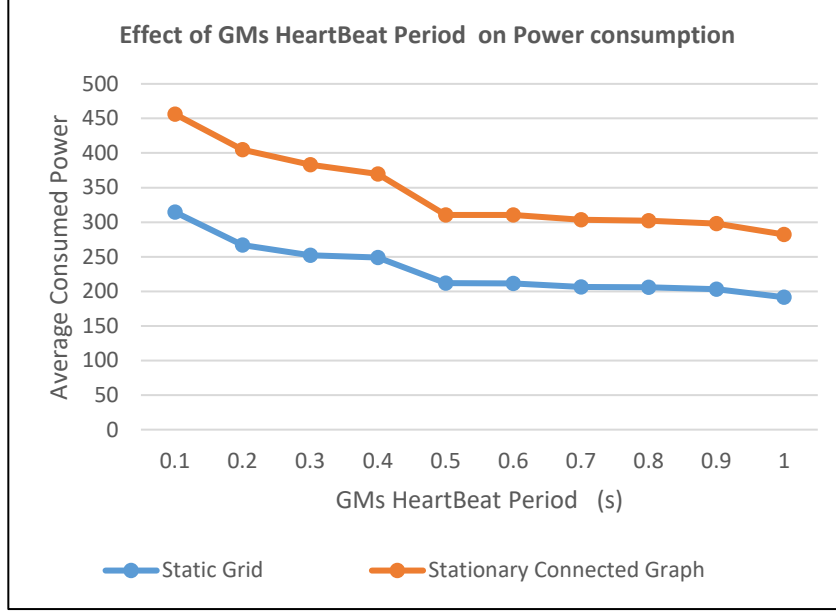


Figure 7-36 The effect of changing $T_{HeartBeatGM}$ on power consumption

From this experiment, we notice that if we could sacrifice a little bit in the connectivity side in favor of reducing power consumption and the overhead, then we can set $T_{HeartBeatGM}$ to 1s.

7.4.2.2.5 $T_{HeartBeatGO}$

This parameter is a management parameter as $T_{HeartBeatGM}$. Thus, we are interested here in showing the connectivity (PM and Connected Components only), management overhead, and power consumption. $T_{HeartBeatGO}$ defines the period that a GO waits before sending a heartbeat message to its GMs. Recall that the proxy assignments are sent using the heartbeat messages, thus the inability of the GO to send at least one heartbeat message would result in a hole in the coverage; If $T_{HeartBeatGO}$ is greater than $T_{pxAssignment}$ that could happen. In this experiment, $T_{pxAssignment}$ is set to 4s and changed $T_{HeartBeatGO}$ from 2s to 6s by increasing 1s to capture its effect the performance. Figure

7-37 shows the effect of this parameter on connectivity. We notice from the figure that when $T_{HeartBeatGO}$ is set to 2s we get the best connectivity results in terms of PM count and number of connected components. Increasing such a value degrades connectivity, specially beyond 4s. If we correlate these graphs with the fact that the numbers of GOs that we get in this experiment were 7.2 and 6.4 for grid and connected graph topologies, respectively, we can conclude that each group were disconnected from the other after passing the 4s mark. Thus, we should keep the value of $T_{HeartBeatGO}$ as low as possible.

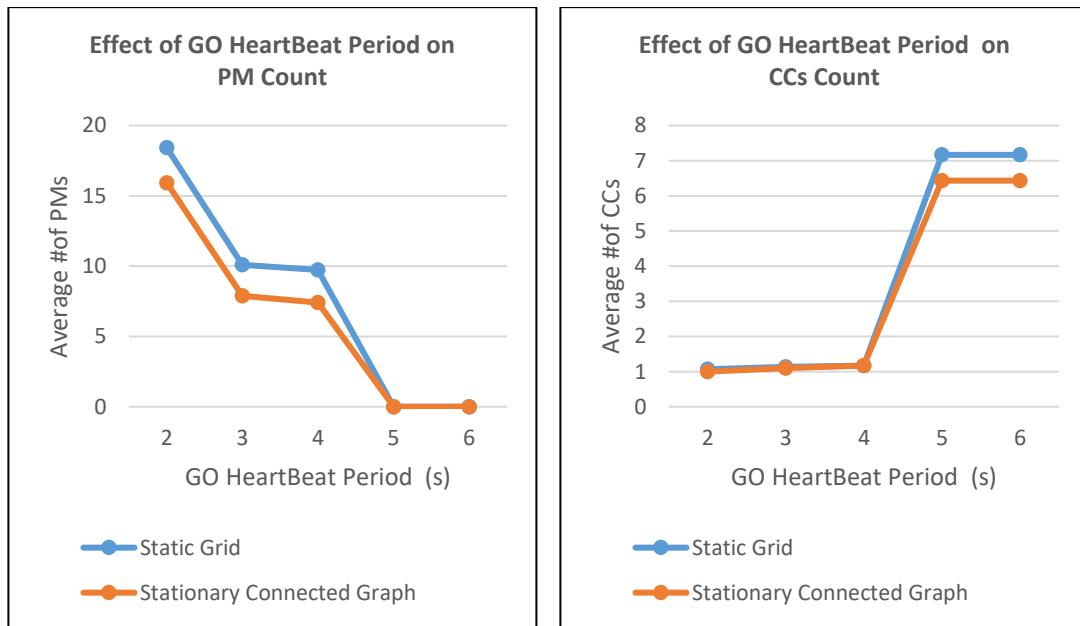


Figure 7-37 The effect of changing $T_{HeartBeatGO}$ on connectivity

From Figure 7-38 and Figure 7-39, we find that the protocol overhead as well as the power consumption are high with lower values of $T_{HeartBeatGO}$, and start to decrease when $T_{HeartBeatGO}$ grows. That is consistent with the fact that the number of responses from the GO decreases with higher values of this parameter, thus reducing the power consumption.

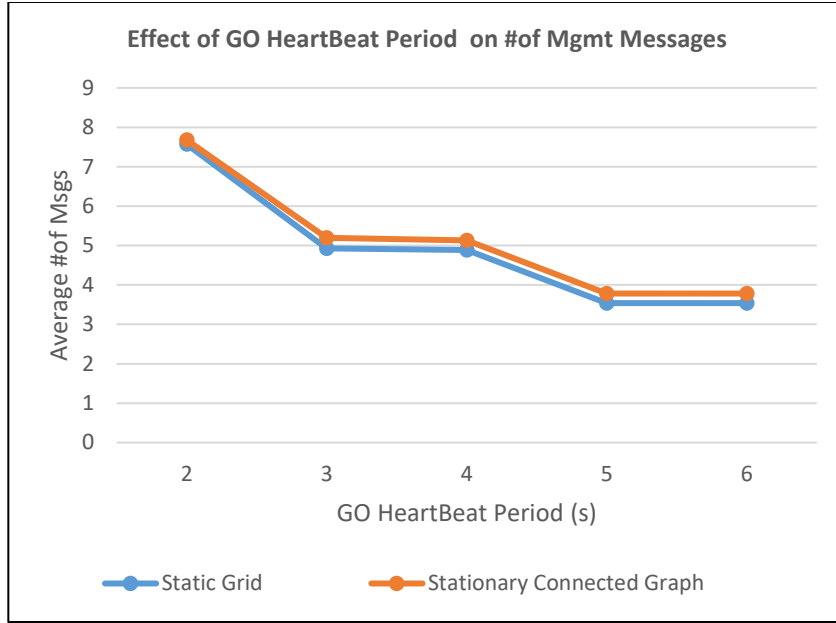


Figure 7-38 The effect of changing $T_{HeartBeatGO}$ on overhead

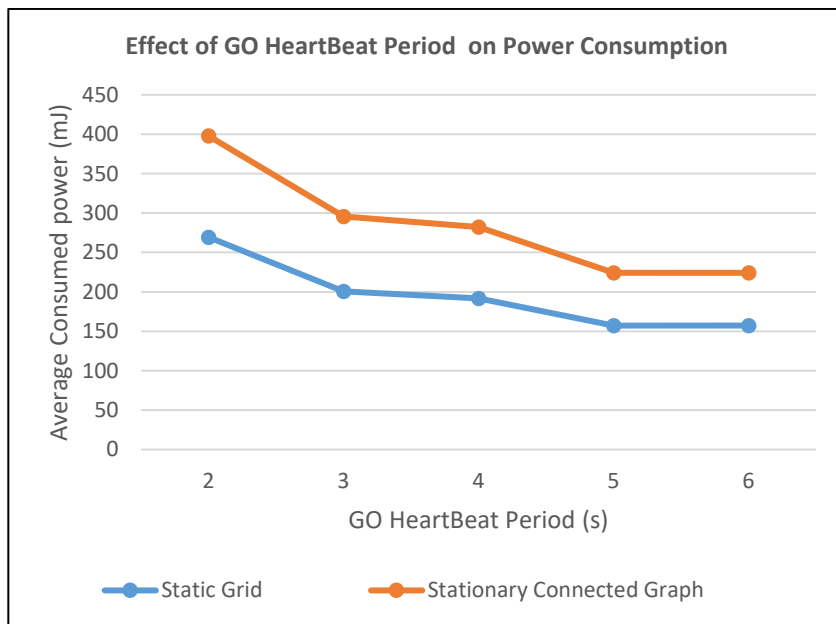


Figure 7-39 The effect of changing $T_{HeartBeatGO}$ on power consumption

As a conclusion from this experiment, we should set $T_{HeartBeatGO}$ to a value that is less than the $T_{pxAssignment}$ time to give the GO the chance to send PM assignments to its

members. A value of 2s gives the best results at the expense of having slightly higher power consumption and overhead.

7.4.2.2.6 $T_{pxAssignment}$

In this experiment, we varied $T_{pxAssignment}$ from 4s to 19s. The minimum value for $T_{pxAssignment}$, which is 4s, allows one GO heartbeat message to reach the GMs. Increasing this value enables more heartbeat messages to be sent. What we have noticed during this experiment is that changing such value beyond 4s has no benefit at all, as the connected components stays at the same value, which is 1.16667, for both topologies. No other performance metric is dependent on $T_{pxAssignment}$.

7.5 Conclusions

In this chapter, we have discussed the development of a simulator for Wi-Fi Direct and the implementation of our integrated suite of protocols (ELN, ADS, EMC, ISNP). Two different set of experiments were performed, one to capture the performance of the Integrated EMC and the other to test the effect of various parameters on performance. The simulation results have shown that our approach can provide connectivity with minimal effect on the power consumption, response time, and overhead. The results have also provided guidelines on how to set the different parameters to yield the best performance.

Chapter 8: Conclusions and Future Work

Advances in communication technology have made data sharing part of daily activities and enabler for many applications. However, the tight coupling between communication links and infrastructure makes it uneasy to share data in certain situations where the infrastructure is down or unavailable. Our focus in this dissertation is on enabling infrastructure-less data sharing between smart devices through the development of a framework that creates and manages P2P links between these devices. This chapter summarizes the contribution of the dissertation and outlines the planned future work.

8.1 Summary of Contribution

In this dissertation, we tackle the problem of data sharing between devices without relying on communication infrastructures by utilizing the D2D communications technologies available on smart devices. We have developed three novel protocols to allow the sharing of data among users using Wi-Fi Direct, a protocol for IP subnet negotiation, and a simulator for Wi-Fi Direct. The following is a summary of the specific research contributions:

A. Alert Dissemination Protocol Using Service Discovery in Wi-Fi Direct (ADS):

ADS is meant for sharing small chunks of data or alerts in a quick manner that is fast and is not limited by a group boundary. ADS uses service discovery on Wi-Fi Direct to exchange data between smart devices without requiring setting up any groups. The devices use the service discovery records to store data

locally. Other devices use service discovery requests to obtain such stored data.

The approach also manages the forwarding of new data and pruning of old data.

- B. *An Efficient and Lightweight Protocol for P2P Networking Smart Devices over Wi-Fi Direct (ELN)*: ELN is a solution for sharing large amounts of data between a small group of devices. ELN utilizes Wi-Fi Direct to setup a group that allows all users to share data with each other. ELN provides a group management solution that manages the addition and removal of devices as well as the required connections.
- C. *Efficient Multi-Group Formation and Communication Protocol for Wi-Fi Direct (EMC)*: This solution targets the case of sharing data among large number of users that span wide area in a power efficient way. EMC dynamically creates Wi-Fi Direct groups of Android smart devices based on certain criteria. EMC then interconnects the formed groups using relay devices to achieve large scale data sharing. Such an approach utilizes ADS for distributing vital protocol specific data and ELN for intra-group interactions.
- D. *IP Subnet Negotiation Protocol for Seamless Multi-Group Communications (ISNP)*: ISNP is developed to overcome the limitation of the Wi-Fi Direct implementation in Android that forces all the created groups to share the same IP subnet, which leads to IP address collisions. By overcoming such limitation, we provide the necessary support to have full inter-group connectivity at the transport layer. ISNP has an application layer module that is integrated with

EMC to allow groups to negotiate their subnets. An OS module is developed to allow the devices to force Android to use their proposed subnets.

- E. *A Simulator for Wi-Fi Direct*: Due to the lack of availability of simulation environments for Wi-Fi Direct, we have developed a simulator to fill such a gap. The simulator utilizes OMNet++ that provides a powerful simulation kernel, INET Framework that have implementation for several networking aspects, and Google OR-Tools that provides linear assignment and connectivity libraries.

The performance of ELN, ADS, EMC, and ISNP is validated through implementation on Android devices and through simulation. In addition, an extensive analysis of the performance of such approaches has been carried out. The results have confirmed the advantages of our protocols in terms of connectivity, response times, protocol overhead, and power consumption.

We envision our proposed protocols to be part of Android and other platforms to facilitate P2P data sharing. Building communication links only is not sufficient to enable the required data sharing, thus a routing mechanism for data is warranted. Witnessing data breaches happening every day elevates the importance of user security and privacy. In addition, relying on Wi-Fi Direct solely to perform communication may not suffice; instead multiple varying technologies could be blended together to provide a sophisticated solution for infrastructure-less data sharing. All these aspects are part of our future-plan, as we highlight next.

8.2 Future Work

As we pointed out, the aim of this dissertation research is to enable users to share data in an infrastructure-less manner. Our investigations have pointed out that Wi-Fi Direct is the most suitable means for building communication links. To that end, we have developed ELN, ADS, EMC, and a supporting protocol ISNP presented in chapters 3, 4, 5, and 6, respectively. In the future, we plan to further to extend our work by providing a data routing service that eases the data retrieval and exchange. In addition, we plan to protect the P2P services against attacks and ensure user privacy through adding several security measures that prevent unauthorized users from cheating or stealing sensitive user information. We plan also to investigate incorporating other technologies, like Bluetooth, in our data sharing solution to provide fault tolerance for communication links. Finally, expanding our work to other platforms, like Apple iOS, is planned in order to support a wide base of users. The following discusses the planned future research activities.

8.2.1 Routing Data Between Groups

Our work provides the necessary means for creating communication links between devices to allow data sharing. Sharing data between group members is handled by the group owner. However, having several connected groups, means that we need to route data once we cross the group boundary. Each group has several PMs to connect it with other groups and selecting one of them to forward the data should be handled to guarantee successful and efficient data sharing. Thus, we plan to develop a routing mechanism that allows forwarding the data to devices on other groups efficiently.

8.2.2 Secure Data Sharing Between Devices

To ensure user privacy and counter attacks we plan to incorporate security measures. Several parts of our work rely on negotiating certain roles between devices through service discovery frames in Wi-Fi Direct. These frames are transmitted using plain text and could be captured by any nearby device. In addition, a malicious device could send crafted service discovery frames to nearby devices to get them to assume that it has the best rank. Such device could then take GO responsibility and intercept any data transferred through the group. Likewise, if a denial of service attack is desired, a malicious device could convince a GO to assign it the PM role, then such device would drop any frame forwarded to it. Thus, we plan to apply several authentication and data integrity mechanisms to prevent unauthorized user from capturing, manipulating, or dropping shared data.

8.2.3 Incorporating Other D2D technologies

We would like to explore the inclusion of other D2D technologies such as Bluetooth Low Energy to increase the robustness of our data sharing framework, especially when the wireless channels are subject to varying levels of interference. Such an extension would also allow devices without Wi-Fi Direct support to involve in the data sharing process. In addition, supporting other technologies can speed up the distribution of data between close by devices, as we will have more than one transceiver to carry the traffic.

8.2.4 Extend our Work to Other Platforms

We hope to have our data sharing solution utilized by a large base of users. Thus, we plan to include support for other platforms, like Apple iOS. Apple has introduced

MultiPeerConnectivity Framework [86] to iOS since version 7.0. Such framework supports the discovery of services provided by nearby devices. It also supports communicating with devices that have such discovered services through different ways, such as messages and streaming. In iOS, the framework uses Wi-Fi networks, Wi-Fi Direct, and Bluetooth for the underlying transport. We plan to utilize such framework for providing a cross-platform support for our data sharing solution.

References

- [1] Betts, B.; et al., "Improving situational awareness for first responders via mobile computing. NASA Ames Research Center," *Smart Systems Research Laboratory*, 2006.
- [2] Boddhu, S. ; et al., "Increasing situational awareness using smartphones", *Proc. SPIE 8389, Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR III*, 83891J (May 1, 2012)
- [3] Foresti, G.; Farinosi, M.; Vernier, M., "Situational awareness in smart environments: socio-mobile and sensor data fusion for emergency response to disasters." *Journal of Ambient Intelligence and Humanized Computing*, April 2015, Volume 6, Issue 2, pp 239-257.
- [4] Mittelstädt, S.; et al., "An Integrated In-Situ Approach to Impacts from Natural Disasters on Critical Infrastructures," in *System Sciences (HICSS), 2015 48th Hawaii International Conference on* , vol., no., pp.1118-1127, 5-8 Jan. 2015.
- [5] Salfinger, A.; et al., "Crowd-Sensing Meets Situation Awareness: A Research Roadmap for Crisis Management," in *System Sciences (HICSS), 2015 48th Hawaii International Conference on* , vol., no., pp.153-162, 5-8 Jan. 2015.
- [6] Huang, Q.; Xiao, Y., "Geographic situational awareness: mining tweets for disaster preparedness, emergency response, impact, and recovery." *ISPRS International Journal of Geo-Information*, vol. 4, issue 3, pp. 1549-1568, 2015.
- [7] Singhal, V., Jha, A., Gairola, A., "A networking solution for disaster management to address liaison failures in emergency response." *Risk Analysis IX* 47, 401, 2014.
- [8] Vieweg, S.; et al., "Microblogging during two natural hazards events: what twitter may contribute to situational awareness." *Proceedings of the SIGCHI conference on human factors in computing systems. ACM*, pp. 1079-1088, 2010.
- [9] Yin, J.; et al., "Using social media to enhance emergency situation awareness." *IEEE Intelligent Systems*, vol. 27, issue 6, pp.52-59, 2012.
- [10] Haddawy, Peter, et al., "Situation awareness in crowdsensing for disease surveillance in crisis situations." *Proceedings of the Seventh International Conference on Information and Communication Technologies and Development (ICTD 2015)*. 2015.
- [11] Emergency AUS application: <http://emergencyaus.info>
- [12] UN-ASIGN application: <https://assign.cern.ch>
- [13] Thompson, C.; et al. "Using smartphones to detect car accidents and provide situational awareness to emergency responders." *Mobile Wireless Middleware, Operating Systems, and Applications. Springer Berlin Heidelberg*, pp. 29-42, 2010.
- [14] Predic, B.; Stojanovic, D.; "Enhancing driver situational awareness through crowd intelligence." *Expert Systems with Applications*, vol. 42, issue. 11, pp. 4892-4909, July 2010.

- [15] Beattie, D.; et al., "What's around the corner?: enhancing driver awareness in autonomous vehicles via in-vehicle spatial auditory displays." *Proceedings of the 8th nordic conference on human-computer interaction: fun, fast, foundational*. ACM, New York, NY, pp. 189-198, 2014.
- [16] Baines, V.; Padget, J., "A Situational Awareness Approach to Intelligent Vehicle Agents." *Modeling Mobility with Open Data*. Springer International Publishing, 2015, pp. 77-103.
- [17] Wymeersch, H.; et al., "Challenges for cooperative ITS: Improving road safety through the integration of wireless communications, control, and positioning," in *Computing, Networking and Communications (ICNC), 2015 International Conference on* , vol., no., pp.573-578, 16-19 Feb. 2015.
- [18] Moradi-Pari, E.; Tahmasbi-Sarvestani, A.; Fallah, Y.P., "A Hybrid Systems Approach to Modeling Real-Time Situation-Awareness Component of Networked Crash Avoidance Systems," in *Systems Journal, IEEE* , vol.PP, no.99, pp.1-10.
- [19] Liu, Z.; Hacigümüs, H.' "Online optimization and fair costing for dynamic data sharing in a cloud data market." *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, pp. 1359-1370, 2014.
- [20] Chen, F.; et al., "Cloud-assisted distributed private data sharing." *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*. ACM, pp. 202-211, 2015.
- [21] Liu, Z.; et al., "Tmds: Thin-model data sharing scheme supporting keyword search in cloud storage." *Information Security and Privacy*. Springer International Publishing, pp. 115-130, 2014.
- [22] Thilakanathan, D.; et al., "A platform for secure monitoring and sharing of generic health data in the Cloud." *Future Generation Computer Systems*, vol.35, pp. 102-113, 2014.
- [23] Liu, Q.; Wang, G.; Wu, J., "Time-based proxy re-encryption scheme for secure data sharing in a cloud environment." *Information Sciences*, vol. 258pp. 355-370, 2014.
- [24] Liu, J.; et al., "Device-to-device communications for enhancing quality of experience in software defined multi-tier LTE-A networks," in *Network, IEEE* , vol.29, no.4, pp.46-52, July-August 2015
- [25] Pratap, A.; Misra, R., "Firefly Inspired Improved Distributed Proximity Algorithm for D2D Communication," in *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International* , vol., no., pp.323-328, 25-29 May 2015
- [26] Jo, M.; Maksymyuk, T.; et al., "Device-to-device-based heterogeneous radio access network architecture for mobile cloud computing," in *Wireless Communications, IEEE* , vol.22, no.3, pp.50-58, June 2015
- [27] Ye, Q.; Al-Shalash, M.; et al., "Resource Optimization in Device-to-Device Cellular Systems Using Time-Frequency Hopping," in *Wireless Communications, IEEE Transactions on* , vol.13, no.10, pp.5467-5480, Oct. 2014

- [28] Lin, X.; Andrews, J.G.; Ghosh, A., "Spectrum Sharing for Device-to-Device Communication in Cellular Networks," in *Wireless Communications, IEEE Transactions on* , vol.13, no.12, pp.6727-6740, Dec. 2014
- [29] Nishiyama, H.; Ito, M.; Kato, N., "Relay-by-smartphone: realizing multihop device-to-device communications," in *Communications Magazine, IEEE* , vol.52, no.4, pp.56-65, April 2014
- [30] Andreev, S.; et al., "Cellular traffic offloading onto network-assisted device-to-device connections," in *Communications Magazine, IEEE* , vol.52, no.4, pp.20-31, April 2014
- [31] Al-Kanj, L.; Poor, H.V.; Dawy, Z., "Optimal Cellular Offloading via Device-to-Device Communication Networks With Fairness Constraints," in *Wireless Communications, IEEE Transactions on* , vol.13, no.8, pp.4628-4643, Aug. 2014
- [32] Google Inc., "Google Nearby," online <https://developers.google.com/nearby/?hl=en>
- [33] Bluetooth SIG (Hrsg.): "Specification of the Bluetooth System: Covered. Core Package version: 4.2," December, 2014.
- [34] Ortiz, P., "Replacing cellular with WiFi direct communication for a highly interactive, high bandwidth multiplayer game," *MSc. Thesis* [online] Available: <http://hdl.handle.net/1721.1/84864>, MIT 2013.
- [35] Krifa, A.; et al., "BitHoc: A content sharing application for wireless ad hoc networks," in *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on* , vol., no., pp.1-3, 9-13 March 2009
- [36] Bruno, R.; Conti, M.; Gregori, E., "Mesh networks: commodity multihop ad hoc networks," in *Communications Magazine, IEEE* , vol.43, no.3, pp.123-131, March 2005
- [37] Oliveira, L.B.; et al., "Evaluation of peer-to-peer network content discovery techniques over mobile ad hoc networks," in *World of Wireless Mobile and Multimedia Networks, 2005. WoWMoM 2005. Sixth IEEE International Symposium on a* , vol., no., pp.51-56, 13-16 June 2005
- [38] Tsai, T.; Chen, J., "IEEE 802.11 MAC protocol over wireless mesh networks: problems and perspectives," in *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on* , vol.2, no., pp.60-63 vol.2, 28-30 March 2005
- [39] Parata, C.; Scarpa, V.; Convertino, G., "Flex-WiFi: a mixed infrastructure and ad-hoc IEEE 802.11 network for data traffic in a home environment," in *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a* , vol., no., pp.1-6, 18-21 June 2007
- [40] Armenia, S.; et al., "Transmission of VoIP traffic in multihop ad hoc IEEE 802.11b networks: experimental results," in *Wireless Internet, 2005. Proceedings. First International Conference on* , vol., no., pp.148-155, 10-14 July 2005

- [41] Bensaou, B.; Fang, Z., "A Fair MAC Protocol for IEEE 802.11-Based Ad Hoc Networks: Design and Implementation," in *Wireless Communications, IEEE Transactions on* , vol.6, no.8, pp.2934-2941, August 2007
- [42] Wi-Fi Alliance, "P2P Technical Group, Wi-Fi Peer-to-Peer (P2P) Technical Specification v1.2," December 2011.
- [43] Camps-Mur, D.; Garcia-Saavedra, A.; Serrano, P., "Device-to-device communications with Wi-Fi Direct: overview and experimentation," in *Wireless Communications, IEEE* , vol.20, no.3, pp.96-104, June 2013.
- [44] IEEE Standard for Information technology--Telecommunications and information exchange between systems Local and metropolitan area networks--Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)* , vol., no., pp.1,2793, March 29 2012.
- [45] R. Motta and J. Pasquale, "Wireless P2P: Problem or opportunity," *Proceedings of the Second IARIA Conference on Advances in P2P Systems*, Florence, Italy, October 2010, pp. 32–37.
- [46] Conti, M.; et al., "Experimenting opportunistic networks with WiFi Direct," *Proceedings of the Sixth Wireless Days Conference*, Valencia, Spain, November 2013, pp. 1-6
- [47] Je, Huigwang; et al., "Mobile network configuration for large-scale multimedia delivery on a single WLAN," *Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific*, vol., no., pp.1-6, Sept. 2014
- [48] Lombera, I.; et al., "Peer management for iTrust over Wi-Fi Direct," *Proc. International Symposium on Wireless Personal Multimedia Communications*, Atlantic City, NJ, Jun. 2013
- [49] Park, J.; et al., "DirectSpace: A Collaborative Framework for Supporting Group Workspaces over Wi-Fi Direct," *MUSIC 2013*, pp. 55-61
- [50] Botrel Menegato, Urbano, et al. "Dynamic clustering in wifi direct technology." Proc. of the 12th ACM international symposium on Mobility management and wireless access (MOBIWAC 2014), Montreal, Canada, May 2014.
- [51] P. Chaki, M. Yasuda and N. Fujita, "Seamless Group Reformation in WiFi Peer to Peer network using dormant backend links," Proc. of the 12th Annual IEEE Consumer Communications and Networking Conference (CCNC 2015), Las Vegas, NV, Jan 2015.

- [52] Zhang, H.; Wang, Y.; Tan, C., "WD2: an improved wifi-direct group formation protocol," *In Proceedings of the 9th ACM MobiCom workshop on Challenged networks (CHANTS '14)*. ACM, New York, NY, USA, 55-60, 2014.
- [53] Duan, Y.; et. al, "Wi-Fi Direct Multi-group Data Dissemination for Public Safety," *Proc. of the World Telecommunications Congress (WTC 2014)*, Berlin, Germany, June 2014.
- [54] A. Laha, X. Cao, W. Shen, X. Tian and Y. Cheng, "An energy efficient routing protocol for device-to-device based multihop smartphone networks," *Proc. of the IEEE International Conference on Communications (ICC 2015)*, London, June 2015.
- [55] W. R. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on Wireless Communications*, Vol. 1, pp. 660-670, 2002.
- [56] Wong, P.; et al., "Automatic Android-based Wireless Mesh Networks." *Informatica (Slovenia)* , 38(4), 2014.
- [57] Doukha, Z.; Moussaoui, S.; Haouari, N., "An efficient alert dissemination protocol in a vehicular ad hoc network," *Digital Information Management (ICDIM), 2012 Seventh International Conference on* , vol., no., pp.68,72, 22-24 Aug. 2012.
- [58] Rehman, O.M.H.; Bourdouden, H.; Ould-Khaoua, M., "Efficient alert messages dissemination in VANETs using single-hop distributed protocols," *Wireless and Mobile Networking Conference (WMNC), 2013 6th Joint IFIP* , vol., no., pp.1,4, 23-25 April 2013.
- [59] Suriyapaiboonwattana, K.; Pornavalai, C.; Chakraborty, G., "An adaptive alert message dissemination protocol for VANET to improve road safety," *Fuzzy Systems, 2009. FUZZ-IEEE 2009. IEEE International Conference on* , vol., no., pp.1639,1644, 20-24 Aug. 2009
- [60] Fogue, M.; Garrido, P.; Martinez, F.J.; Cano, J.; Calafate, C.T.; Manzoni, P., "An Adaptive System Based on Roadmap Profiling to Enhance Warning Message Dissemination in VANETs," *Networking, IEEE/ACM Transactions on* , vol.21, no.3, pp.883,895, June 2013
- [61] Garcia-Lozano, E.; Campo, C.; Garcia-Rubio, C.; Cortes-Martin, A.; Rodriguez-Carrion, A.; Noriega-Vivas, P. "A Bandwidth-Efficient Service for Local Information Dissemination in Sparse to Dense Roadways," *Sensors* **2013**, 13, 8612-8639.
- [62] Sardar, A., "Improving Performance of IEEE 802.11 p MAC Layer for Emergency Message Dissemination," *MS Thesis*, Tamere University of Technology, Finland, 2013.

- [63] Huang, H.; et al., "Performance Evaluation of an Alert Dissemination Engine based on the AT&T Enterprise Messaging Network."
- [64] Chen, Yu-Jia; Lin, Chia-Yu; Wang, Li-Chun, "A personal emergency communication service for smartphones using FM transmitters," *Personal Indoor and Mobile Radio Communications (PIMRC), 2013 IEEE 24th International Symposium on* , vol., no., pp.3450,3455, 8-11 Sept. 2013.
- [65] Chen, Yu-Jia; Lin, Chia-Yu; Wang, Li-Chun, "Sensors-assisted rescue service architecture in mobile cloud computing," *Wireless Communications and Networking Conference (WCNC), 2013 IEEE* , vol., no., pp.4457,4462, 7-10 April 2013
- [66] Teranishi, Y.; Shimojo, S., "MONAC: SNS message dissemination over smartphone-based DTN and cloud," *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on* , vol., no., pp.158,159, Aug. 31 2011-Sept. 2 2011
- [67] Thomas J.; Robble, J., "Off grid communication with Android: Meshing the mobile world," <https://media.blackhat.com/eu-13/briefings/Thomas/bh-eu-13-off-grid-communication-wp.pdf>
- [68] Kolios, P.; et al., "Qualifying explore and exploit for efficient data dissemination in emergency adhoc networks," *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on* , vol., no., pp.303,307, 24-28 March 2014.
- [69] Jalihal, D.; et al., "A rapidly deployable disaster communications system for developing countries," *Communications (ICC), 2012 IEEE International Conference on* , vol., no., pp.6339,6343, 10-15 June 2012
- [70] George, S.M.; et al., "DistressNet: a wireless ad hoc and sensor network architecture for situation management in disaster response," *Communications Magazine, IEEE* , vol.48, no.3, pp.128,136, March 2010
- [71] Lorincz, K.; et al., "Sensor networks for emergency response: challenges and opportunities," *Pervasive Computing, IEEE* , vol.3, no.4, pp.16,23, Oct.-Dec. 2004
- [72] Lien, Yao-Nan; Jang, H; Tsai, T., "A MANET Based Emergency Communication and Information System for Catastrophic Natural Disasters," *Distributed Computing Systems Workshops, 2009. ICDCS Workshops '09. 29th IEEE International Conference on* , vol., no., pp.412,417, 22-26 June 2009

- [73] Reina, D.G.; et al., "An Evolutionary Computational Approach for Optimizing Broadcasting in Disaster Response Scenarios," *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, 2013 Seventh International Conference on , vol., no., pp.94,100, 3-5 July 2013
- [74] Reina, D. G.; et al., "Multi-objective performance optimization of a probabilistic similarity/dissimilarity-based broadcasting scheme for mobile ad hoc networks in disaster response scenarios." *Soft Computing* (2013): 1-12.
- [75] Qualcomm Technologies, Inc., "LTE Direct: The Case for Device-to-Device Proximate Discovery," Feb. 2013, <http://www.qualcomm.com/media/documents/qualcomm-research-ltedirect-overview>.
- [76] Shahin, A.; Younis, M., "A framework for P2P networking of smart devices using Wi-Fi direct," in 2014 IEEE 25th International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC): Services Applications and Business (IEEE PIMRC 2014 - Services Applications and Business), Washington, DC, USA, Sep. 2014, pp. 2087–2092.
- [77] Shahin, A., "WiFi Direct Sensors," [online] Available: <https://play.google.com/store/apps/details?id=esnetlab.apps.android.wifidirectsensors>
- [78] Shahin, A., "WiFi Direct Group Chat," [online] Available: <https://play.google.com/store/apps/details?id=esnetlab.apps.android.wifidirect.discovery>
- [79] Shahin, A.; Younis, M., "Alert Dissemination Protocol Using Service Discovery in Wi-Fi Direct," Proc. *IEEE Int'l Symp. Communications Software, Services and Multimedia Applications. (ICC'15)*, London, UK, June. 2015.
- [80] Shahin, A.; Younis, M., "Efficient Multi-Group Formation and Communication Protocol for Wi-Fi Direct," Proc. *40th Annual IEEE Conference on Local Computer Networks (LCN 2015)*, pp. 442-445, Clearwater Beach, USA, Oct. 2015.
- [81] Shahin, A.; Younis, M., "Wi-Fi Direct based Peer-to-Peer System for Smart Devices," to be submitted to *IEEE Systems Journal*.
- [82] J. Munkres "Algorithms for the Assignment and Transportation Problems" J. Society for Industrial and Applied Math. 5(1), Mar. 1957.
- [83] Andras Varga et al. "OMNet++ Discrete Event Simulator," [online] Available: <https://omnetpp.org/>

- [84] Andras Varga et al. "INET Framework," [online] Available: <https://inet.omnetpp.org/>
- [85] Google, "OR-Tools," [online] Available: <https://developers.google.com/optimization/>
- [86] Apple, "Apple Developer Documentation," [online] Available: <https://developer.apple.com/reference/multipeerconnectivity>

