António Gelásio Frazão Isidro Teófilo

Mestre em Engenharia Electrotécnica e de Computadores

WiFi-Direct InterNetworking

Dissertação para obtenção do Grau de Doutor em

Informática

Orientador: Hervé Miguel Cordeiro Paulino,
Professor Auxiliar,
Universidade NOVA de Lisboa

Co-orientador: João Manuel dos Santos Lourenço,
Professor Auxiliar,
Universidade NOVA de Lisboa

Júri

Presidente:	Professor Doutor José Augusto Legatheaux Martins
Arguentes:	Professor Doutor Luciano Baresi
	Professor Doutor Nuno Miguel Carvalho dos Santos
Vogais:	Professor Doutor José Augusto Legatheaux Martins
	Professor Doutor Rolando da Silva Martins
	Professor Doutor Hervé Miguel Cordeiro Paulino

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Agosto, 2019

WiFi-Direct InterNetworking

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

To Ana.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisers Prof. Hervé Miguel Cordeiro Paulino and Prof. João Manuel dos Santos Lourenço for the continuous support of my Ph.D research, for their patience, motivation and knowledge.

Besides that, I would like to thank the remaining members of the thesis committee: Prof. José Augusto Legatheaux Martins, Prof. Luciano Baresi, Prof. Nuno Miguel Carvalho dos Santos and Prof. Rolando da Silva Martins, for their insightful comments and encouragement, but also for the hard questions, which incite me to widen my research from various perspectives.

I would also like to thank, once again, Prof. José Augusto Legatheaux Martins and Prof. Rolando da Silva Martins, as members of the Thesis Accompanying Committee, for their, once again insightful comments and encouragement, but also for the hard questions, which incite me to widen my research from various perspectives

I would also like to thank the Hyrax Project (grant CMUP-ERI/FIA/0048/2013), the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS) and the Department of Informatics (DI) of NOVA School of Science and Technology (FCT) for the partial support for the research activities.

I would also like to thank Prof. Nuno Correia (PhD Coordinator) and Prof. Luís Caires (Department chair) for their help in administrative processes in very stressed moments.

I would also like to thank all in DI and in particular the reading group in Computer Systems for the fresh, creative, constructive and motivating environment. In special I would like to thank Prof. Nuno Preguiça and Prof. João Leitão.

I would also like to thank, again, Prof. Nuno Preguiça as my initial PhD tutor.

I would also like to thank my PhD colleagues: João Silva, Diogo Remédios, Valter Balegas and Paulo Araújo, which in a way or another they contributed to create a nice and motivating research environment.

I would also like to thank all that contributed to keep the DI cluster of machines running and updated. Without their precious support, I would not have been able to conduct this research.

I would also like to thank, again, Diogo Remédios for the initial joint work. That was a very enjoyable and important step in the research.

I would also like to thank all in the Departamento de Engenharia Eletrónica e Telecomunicações e de Computadores (DEETC) of Instituto Superior de Engenharia de Lisboa

(ISEL) of Instituto Politécnico de Lisboa (IPL) that contributed to this PhD. In particular, I would like to thank: Diogo Remédios (again x2), Porfírio Filipe, Carlos Gonçalves, Pedro Fazenda and Jorge Pais for their sacrifice to handle part of my work, which allowed me to focus on the PhD. Without their help I doubt that I could have finished the PhD within the deadlines. I would also like to thank Pedro Mendes Jorge (course coordinator of my teaching activities) for the class timetables, at ISEL, which enabled me to focus on the PhD.

I would also like to thank Rosário Ainslie for the help with the English.

I would also like to thank my friends for their patience and unconditional support.

Finally, my deep and sincere gratitude to my family, specially to Ana, for their continuous love, help, patience and support.

"Begin at the beginning," the King said, very gravely,
"and go on till you come to the end: then stop."
– Lewis Carroll, Alice in Wonderland

Abstract

We are on the verge of having ubiquitous connectivity. However, there are still scenarios where public communication networks are not reachable, are saturated or simply cannot be trusted. In such cases, our mobile phones can leverage device-to-device communication to reach the public network or to enable local connectivity.

A device-to-device communication technology, with at least WiFi speed and range, will offer sufficient connectivity conditions for interconnection in areas/situations where it is not currently possible. Such advance will foster a new breed of systems and applications. Their widespread adoption is, nonetheless, bound to their usage in off-the-shelf devices. This raises a problem because the device-to-device communication technologies currently available in off-the-shelf mobile devices have several limitations: Bluetooth is limited in speed and range, Wi-Fi Direct is limited in speed and connectivity for medium and large scenarios, and WiFi-Aware is a new and untested technology, whose specification does not cover large scenarios.

In this thesis, we address this problem by presenting two communication topologies and a network formation algorithm that enable the use of Wi-Fi Direct communication between off-the-shelf mobile devices in medium and large scale scenarios. The communication topologies, named Group-Owner Client-Relay Group-Owner and Group-Owner Group-Owner, allow for Wi-Fi Direct intergroup communication, whilst the network formation algorithm, named *RedMesh*, systematically creates networks of Wi-Fi Direct groups. The algorithm proved to be very effective, achieving full connectivity in 97.28% of the 1 250 tested scenarios.

The *RedMesh* algorithm distinguishes itself for being the first one to use Wi-Fi Direct communication topologies that can form tree and mesh structures, and for being the first algorithm able to build networks that can rely only on unicast communication. We may hence conclude that the work developed in this thesis makes significant progress in the formation of large scale networks of off-the-shelf mobile devices.

Keywords: Wireless communication, Wi-Fi Direct, Autonomous networks, network formation algorithms, mesh networks

Resumo

Correntemente, estamos à beira de ter conectividade de forma ubíqua. Contudo ainda há cenários que as redes públicas de comunicação não cobrem, ou que não podem ser utilizadas por estarem saturadas ou por falta de confiança no seu uso. Em tais situações, os nosso telemóveis poderiam utilizar comunicação dispositivo-a-dispositivo para chegar à rede pública ou permitir comunicação local.

Uma tecnologia de comunicação dispositivo-a-dispositivo com a velocidade e o alcance do WiFi iria oferecer uma satisfatória conectividade em tais condições e permitir o aparecimento de um novo tipo de sistemas e aplicações. No entanto, a sua ampla adoção somente será garantida caso a sua utilização seja permitida nos telemóveis sem serem alterados (com a garantia intacta). Contudo, as tecnologias de comunicação dispositivo-a-dispositivo, disponíveis para esses dispositivos, têm varias limitações: o Bluetooth é limitado em velocidade e alcance; o Wi-Fi Direct é limitado em velocidade e conectividade para cenários de media larga escala e o WiFi-Aware é uma tecnologia por testar mas que a sua especificação não cobre cenários de larga escala.

Nesta tese nós abordamos esse problema apresentando duas topologias de comunicação, chamadas de Group-Owner Client-Relay Group-Owner e Group-Owner Group-Owner, que permitem a comunicação entre grupos de dispositivos Wi-Fi Direct e um algoritmo, chamado de *RedMesh*, que permite a formação sistemática de redes pela interligação desses grupos. O algoritmo provou a sua eficiência ao atingir 97.28% de sucesso em conseguir conectividade entre todos os nós em 1250 cenários.

O algoritmo *RedMesh* diferencia-se por ser o primeiro a utilizar conjuntamente topologias de comunicação Wi-Fi Direct que permitem a formação de estruturas do tipo árvore e malha e também por construir redes que permitem a utilização exclusiva de comunicação por *unicast*. Tal leva-nos a concluir que o trabalho desenvolvido nesta tese faz significativos avanços na área das redes em larga escala em dispositivos móveis não alterados.

Palavras-chave: Comunicação em redes móveis, Wi-Fi Direct, redes autónomas, algoritmos de formação de redes, redes do tipo malha

Contents

List of Figures xvii

List of Tables xix

Acronyms xxi

1 Introduction 1
- 1.1 Context . . . 1
- 1.2 Challenges and statements . . . 3
- 1.3 Proposed solutions . . . 6
- 1.4 Contributions . . . 7
- 1.5 Document structure . . . 8

2 WiFi-Direct Group Interconnection 9
- 2.1 Context and problems . . . 9
  - 2.1.1 WiFi-Direct introduction . . . 9
  - 2.1.2 Group interconnection challenges . . . 12
  - 2.1.3 Group interconnection current solutions . . . 12
  - 2.1.4 Conclusion . . . 15
- 2.2 Solutions and results . . . 15
  - 2.2.1 Proposed group interconnection topologies . . . 15
  - 2.2.2 Topologies analysis . . . 20
  - 2.2.3 Topologies evaluation . . . 32
  - 2.2.4 Conclusion . . . 36
- 2.3 Conclusions . . . 36

3 WiFi-Direct Network Formation 39
- 3.1 Context and problems . . . 39
  - 3.1.1 Existing WiFi-Direct network formation algorithms . . . 39
  - 3.1.2 Network formation algorithms from BSF and MANETS . . . 41
  - 3.1.3 Conclusion . . . 43
- 3.2 Solutions and results, the RedMesh algorithm . . . 43
  - 3.2.1 Algorithm preliminary considerations . . . 43

3.2.2 Algorithm introduction and stages 49
3.2.3 Algorithm auxiliary procedures 55
3.2.4 Algorithm evaluation 60
3.2.5 Algorithm limitations 75
3.2.6 Reflections about network redundancy and node churn 78
3.2.7 Conclusion 80
3.3 Conclusions 80

4 Conclusion **83**
4.1 Q1 – WFD intergroup for large scale mesh networks 83
4.2 Q2 – WFD large scale mesh network formation 84
4.3 MQ – WFD large scale mesh networks 84
4.4 Future work and open issues 84

Bibliography **87**

List of Figures

1.1 Scenarios that can require autonomous connectivity 4

2.1 A 3 group scenario in GOCR topology. 14
2.2 A 3 group scenario in GO2CR topology. 18
2.3 A 3 group scenario in GOCRGO topology. 19
2.4 A 3 group scenario in GOGO topology. 20
2.5 GOCRGO topology applied to a 2D scenario. 25
2.6 GOGO topology applied to a 2D scenario. 25
2.7 GOCRGO and GOGO in a 2D grid scenario. 26
2.8 Application of GOCRGO topology to a sparse scenario. 29
2.9 Application of GOGO topology to a sparse scenario. 29
2.10 A crowded scenario with GOCRGO topology. 30
2.11 A crowded scenario with GOGO topology. 30
2.12 A 3 group scenario in $GOCR_{UC}$ variant. 32
2.13 Experimental scenario for each tested topology. 33
2.14 Topologies communication speed. 34
2.15 Topology consumed energy. 35

3.1 Slave interface connection. 44
3.2 Connectivity scenario. 45
3.3 Scenario modelled in a UDG and in an AG. 46
3.4 An AG modelled scenario not connected by WFD. 48
3.5 Running scenario after DNE stage. 50
3.6 Running scenario after CLB stage. 53
3.7 Running scenario after MCI stage. 55
3.8 Running scenario after FCI stage. 56
3.9 Connectivity after MCI and FCI stages. 63
3.10 Communication results for all stages. 63
3.11 Time results for all stages. 64
3.12 Cluster interconnections in MCI and FCI stages. 65
3.13 MCI interconnection rule applications. 66
3.14 FCI interconnection rule applications. 66
3.15 MCI failed interconnection rule applications. 67

3.16 *RedMesh* with and without one rule. 67
3.17 LCsToGOs results. 68
3.18 Rule order results. 70
3.19 Connectivity with and without intermediate connections. 70
3.20 Scenario not connected with $L = 5$ and connected with $L = 6$. 72
3.21 1^{st} scenario connected with $L = 5$ and not connected with $L = 6$. 72
3.22 2^{nd} scenario connected with $L = 5$ and not connected with $L = 6$. 73
3.23 *RedMesh* versus MAGNET. 75
3.24 Scenarios not connected by *RedMesh*: 1 and 2. 76
3.25 Scenarios not connected by *RedMesh*: 3, 4 and 5. 77
3.26 Scenarios not connected by *RedMesh*: 6 and 7. 78

List of Tables

1.1 Samsung Galaxy S evolution 2

2.1 Communication assessment 16
2.2 Topologies spatial node requirements. 21
2.3 Topologies communication speed. 22
2.4 Topologies routing operations. 23
2.5 Topologies comparison 31
2.6 Experimental setup. 34

3.1 RedMesh cluster interconnection rules. 57
3.2 Scenarios metrics 61
3.3 Results for DNE, CLB and CNG stages 62

Acronyms

A5C	Android 5 Compliant.
AG	Arbitrary Graph.
AP	Access Point.
BAS	Bonjour Advertising Service.
BCF	Broadcast Factor.
BCS	Broadcast Speed.
BLT	Bluetooth.
BSF	Bluetooth Scatternet Formation.
CIP	Cluster Interconnection Procedure.
CL	Client.
CLB	Cluster Building.
CNG	Cluster Neighbourhood Gathering.
CR	Client Relay.
CSS	Candidate Slave Set.
DFS	Depth-First Search.
DHCP	Dynamic Host Configuration Protocol.
DNE	Dominant Node Election.
DTN	Delay Tolerant Network.
EBO	Enslaved By Others Set.
EIP	End of Interconnections Procedure.
FCI	Final Cluster Interconnection.
GO	Group Owner.
GO2CR	Group-Owner 2 Client-Relay.

GOCR	Group-Owner Client-Relay.
$GOCR_{UC}$	Group-Owner Client-Relay UniCasts.
GOCRGO	Group-Owner Client-Relay Group-Owner.
GOD	Dominant Group Owner.
GOGO	Group-Owner Group-Owner.
LC	Legacy Client.
LTE	Long Term Evolution.
MANET	Mobile Ad Hoc Network.
MCF	Multipeer Connectivity framework.
MCI	Main Cluster Interconnection.
MIS	Maximal Independent Set.
NAN	Neighbor Awareness Networking.
NoA	Notice of Absence Protocol.
non-A5C	non-Android 5 Compliant.
OPS	Opportunistic Power Save Protocol.
PSK	Pre-Shared Key.
SGF	Smart-Group-Formation.
SSID	Service Set Identifier.
UDG	Unit-Disk Graph.
WFD	Wi-Fi Direct.
WFR	Wi-Fi Range.
WFS	Wi-Fi Speed.
WiFi	Wi-Fi.
WPA2	Wireless Protected Access - 2.
WPS	Wi-Fi Protected Setup.

Chapter 1

Introduction

The main research statement of this thesis is *"It is possible to build large scale mesh networks of Wi-Fi Direct (WFD) enabled devices"*. This statement is decomposed into two: *"It is possible to do WFD intergroup connection and communication for large scale mesh networks"* and *"It is possible to systematically form WFD large scale mesh networks"*.

This Introduction presents the context and all the steps taken along the research process driven by those statements. It is composed by the following sections: 1.1 Context; 1.2 Challenges and statements; 1.3 Proposed solutions; 1.4 Contributions and 1.5 Document structure.

1.1 Context

According to Ericsson Mobility Report of November 2017 [22], the number of smartphone subscriptions will increase from 4.4 billion in 2017, to 7.3 in 2023. The same report also unveils that smartphone mobile data traffic will increase from 13.8% to 40% of total data traffic, in the same period. Therefore we can state that smartphones will become ubiquitous and users will become huge data producers and/or consumers.

To help this revolution, the capabilities of the hardware of the smartphones continue increasing in every aspect. Table 1.1 shows the evolution of a well known mobile phone model over the last 9 years. It shows an increase of CPU power by more than 8 times, of main memory by 16 times, and of internal memory by 32 times. Battery power and display area have also increased by factors of 2.2 and 2.3, respectively. Furthermore, nowadays we can find mobile phones equipped with many communication technologies (GSM/CDMA/HSPA/LTE, Wi-Fi 802.11 a/b/g/n/ac, dual-band, WiFi-Direct, DLNA,

Table 1.1: Samsung Galaxy S evolution

	Version of Samsung Galaxy S model									
Property:	S	S II	S III	S4	S5	S6	S7	S8	S9	S10
Release year	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
CPU (cores)	1	2	4	8	4	8	8	8	8	8
CPU (GHz)	1.0	1.2	1.4	1.4†	2.5	1.8†	1.95†	2.0†	2.25†	2.24†
RAM (GB)	.5	1	1	2	2	3	4	4	4	8
SDRAM (GB)	16	32	64	64	32	128	64	64	256	512
Battery (Ah)	1.5	1.7	2.1	2.6	2.8	2.6	3.0	3.0	3.0	3.4
Display size (")	4.0	4.3	4.8	5.0	5.1	5.1	5.1	5.8	5.8	6.1

Source: www.gsmarena.com. SDRAM in maximum values. †, in average.

hotspot, Bluetooth, NFC, etc.), sensors (GPS, fingerprint, accelerometer, compass, gyroscope, proximity, ambient light, barometer, heart rate, SpO2, microphones, etc.) and cameras. This hardware increase confirms the trend of the use of mobile devices to become major players as data producers and/or consumers.

Current smartphone capabilities enable rich consumer interactions over the internet, producing and consuming contents like music, photos, videos or data in general. Such collaborative interactions should not be restricted to the internet, as users may demand to have them even when there are no communication infrastructures (for example, in isolated areas and disaster situations), the communication infrastructure is saturated (such us in crowded venues, as in big sports and cultural events [22, 31, 48]) or when the public network cannot be trusted (like in political demonstrations in countries that censor network traffic or in the event of governmental actions in foreign countries) [58].

Collaborative systems, exclusively based on mobile devices, may not only help to disseminate information (like messages and photos) but also to share resources among users. These systems can act as a mobile cloud for data and computation [34], providing users with the sum of all the connected resources. Real-world examples of applications that can be used in such scenarios include:

- the USA military CBMEN (Content-Based Mobile Edge Networking) program[1], aiming at "*enabling efficient, transparent distribution of content in mobile ad hoc network environments*" that will allow soldiers to share information in war scenarios, like in Figure 1.1a[1];

- a photo face recognition system to look for missing people in crowded scenarios, to be used like in Figure 1.1e[2];

- a video sharing system for sport events recorded and watched by in-place fans [12];

[1] http://www.darpa.mil/program/content-based-mobile-edge-networking, accessed on Jul 26, 2019.
[2] https://sverigesradio.se/sida/artikel.aspx?programid=83&artikel=7243712, accessed on Jul 26, 2019.

- a system supporting the exchange of emergency messages in disaster scenarios, to be used like in Figure 1.1c[3];
- a system supporting the exchange of messages and videos for political demonstrations, to be used like in Figure 1.1e[2];
- a system to enable cave explorers/rescuers to communicate and transfer data, among themselves and to the outside team, to be used like in Figure 1.1b[4];
- an educational system to enable data exchange for outdoor classes, to be used like in Figure 1.1f[5]; and
- a system for scientists or photographers, connecting mobile phones, sensors, photo cameras, wireless flashes, external disks, offering a kind of a cloud to save and share data and photos, to be used like in Figure 1.1d[6].

1.2 Challenges and statements

However, the feasibility of such infrastructureless collaborative mobile systems is bound to the ability to interconnect them. In that context, Bluetooth (BLT), Wi-Fi Direct (WFD), WiFi-Aware, MCF and MeshTalk are (or will be) communication technologies that can be used for direct interconnection of off-the-shelf mobile devices. Here we will give a brief description of these technologies.

Before that, we have to clarify that devices may be *non-rooted* or *rooted*. *Non-rooted* devices are the ones which are in the conditions they are sold. Thus, they keep their manufacturer/seller warranty valid. *Rooted* devices are the ones which have their operating system changed outside the manufacturer/seller control and consequently they are out of warranty. *Rooted* devices may incorporate any software, including a complete new operating system. Its purpose is to explore the use of the hardware beyond what the operating system from the manufacturer offers. Hence, *rooted* devices may improve their use, but users usually reject it, as they are not interested in losing the warranty of their phones. Therefore, we only target *non-rooted*, which is off-the-shelf, devices to allow a general use of the intended device-to-device connectivity that we are looking for.

Bluetooth (BLT) [6] is a technology for device-to-device communication, conceived for devices with few capabilities and reduced battery. Therefore, its communication speed and coverage range are much more restrictive than those of Wi-Fi (WiFi) or WFD. Besides that, there is a lot of work in BLT intergroup communication (named Bluetooth Scatternet

[3] https://temblor.net/earthquake-insights/nasa-radar-maps-reveal-massive-extent-of-amatrice-damage-from-italy-earthquake-1271/, accessed on Jul 26, 2019.

[4] https://svhtt.thuathienhue.gov.vn/?gd=6&cn=1&id=187&tc=2109, accessed on Jul 26, 2019.

[5] http://jonnelexplorer.blogspot.com/2014/06/mt-telakawa-capas-national-shrine.html, accessed on Jul 26, 2019.

[6] https://blog.nature.org/science/2019/01/03/what-scientists-can-learn-from-sound-and-silence/, accessed on Jul 26, 2019.

a: Autonomous military scenario

b: Cave scenario

c: Earthquake scenario

d: Scientific expedition scenario

e: Crowded scenario

f: Outdoor educational scenario

Figure 1.1: Scenarios that can require autonomous connectivity

Formation (BSF)) like in [14, 28, 29, 40, 41, 54, 59, 60] and, also, recently BLT version 5 specification endorsed BSF in its Mesh networking Specification [3].

Wi-Fi Direct (WFD) [56] is an interesting technology as it may offer device-to-device communication with WiFi range and speed. It is a WiFi-based protocol that can form groups of devices, where one device named Group Owner (GO) acts as a coordinator of the group, providing advertising and discovery of devices and services, group formation and authentication, Dynamic Host Configuration Protocol (DHCP) and data forwarding services for group members. Nevertheless, as a WFD group can usually support only 8 clients and all of them must be in the GO coverage range, crossing groups is fundamental for medium and large scale scenarios. The WFD specification does not tackle group interconnection, leaving it open for research. Yet, once built, a WFD multi-group communication network may provide long-range and fast data transfer in scenarios with

off-the-shelf mobile devices and with no network infrastructure, paving the way for collaborative systems that can be useful in the situations already mentioned.

WiFi-Aware/Neighbor Awareness Networking (NAN) [1] is a device-to-device technology recently proposed by the WiFi Alliance. It organizes devices in clusters, called *NAN Clusters*, sharing parameters like a cluster *id*, a synchronization clock and a main master (anchor master). Clusters may merge to form a common cluster. Devices may participate in several clusters, but that is not covered by the specification. Devices may operate concurrently in a NAN cluster and also on WiFi infrastructure or WFD, but that is also not covered by the specification. Communication is based on the WiFi 802.11 physical layer complemented with a NAN MAC layer. Google already defined an API for WiFi-Aware in its Android 8.0 version[7] and there are now some mobile phones, like Google Pixel 3, with WiFi-Aware certification. Therefore, we conclude that WiFi-Aware is now ready to be tested. However, it cannot be used in large scale scenarios, as: 1) clusters have a single main (anchor) master, which will not be a scalable solution for large clusters; and 2) neither multi-cluster operation, nor concurrent operation with other technology is defined.

There are also the **Multipeer Connectivity framework (MCF)**[8] developed by Apple. This technology offers device-to-device discovery and communication, but also over the WiFi infrastructure for iOS or over the internet for macOS. There is no public specification and the on-line documentation only affirms that MCF uses peer-to-peer WiFi (Wi-Fi Direct) and BLT for the underlying transport. It organizes devices, called peers, in groups, called sessions. Sessions can have a total maximum of 8 devices. A peer may be connected to several sessions and may transfer data among sessions. A peer can send/receive data to/from another to peer in a common session. Hence, we conclude that MCF builds on Wi-Fi Direct and BLT and should have their identified limitations for large scale scenarios.

MeshTalk is a device-to-device technology recently announced by the Oppo manufacturer[9]. This technology promises to allow device-to-device communication up to 3 Km away, without using cellular networks, WiFi or BLT. They claim that this technology will enable text messages and voices calls, and offer low-power consumption. They also claim that devices will be able to last 72 hours in standby (with MeshTalk active). However, its specification is not available and the technology is, for now, just a promise.

From BLT, WFD, WiFi-Aware, MCF and the possible MeshTalk we can conclude that the mobile reality is moving strongly to enable device-to-device communication and a connectivity solution for large scale is now, or in a near future, a real demand.

From the existing device-to-device technologies, when we had to make the selection, we elected Wi-Fi Direct as it was the one that could provide WiFi range and speed. The

[7] https://developer.android.com/guide/topics/connectivity/wifi-aware, accessed on Jul 26, 2019.
[8] https://developer.apple.com/documentation/multipeerconnectivity, accessed on Jul 26, 2019.
[9] https://www.gsmarena.com/oppo_meshtalk_can_make_calls_and_send_texts_with_no_carrier_or_internet_connection-news-37791.php, accessed on Jul 26, 2019.

WiFi-Aware is now available to be tested, but not ready at that time. Nevertheless, WiFi-Aware needs a solution for large scale scenarios.

Focusing on WFD, the existing work in inter-group communication has several drawbacks as it mainly forms tree like networks, it requires many nodes and transmissions per group and it uses broadcast communication. However, for medium or large scale scenarios, mesh networks offer advantages over tree ones, as they can be built in a distributed way and offer alternative traffic paths, which can reduce the average short path length between devices and enable redundant paths. The main challenges here were: all GOs have the same IP address and devices connected to two groups can only initiate communications through their priority interface.

The meanwhile produced state-of-the-art in network formation algorithms for WFD attempt to build tree networks which are not adequate for large networks or they create mesh networks but with limited connectivity. Besides that, they use communication topologies that use broadcasts, which are slow and energetically demanding. The main challenges here were: to have and select a topology to use; define which nodes to use for interconnection; and how to connect them.

This context motivated the main research question of this thesis:

MQ) *"Is it possible to build large scale mesh networks of WFD enabled devices?"*.

The answer to the main research question (MQ) raised two base questions:

Q1) *"Is it possible to do WFD intergroup connection and communication for large scale mesh networks?"* and

Q2) *"Is it possible to systematically form WFD large scale mesh networks?"*.

However, as a solution using WFD will be the only existing solution for off-the-shelf (*non-rooted*) devices, the main research question can be framed by the following wider question:

WQ) *"Is it possible to build large scale mesh networks of off-the-shelf mobile devices?"*.

1.3 Proposed solutions

To overcome existing challenges, in relation to WFD group interconnection, we developed the Group-Owner Client-Relay Group-Owner (GOCRGO) and Group-Owner Group-Owner (GOGO) connection and communication topologies. They, together, offer complementary ways to cross WFD groups and provide connection for any configuration of nodes. They also accomplish two secondary goals of using only unicast communication and requiring the minimum number of nodes and transmissions per group.

Building from GOCRGO and GOGO topologies, we developed *RedMesh*, a mesh network formation algorithm for off-the-shelf WFD enabled devices. The conceived algorithm defined several cluster interconnection rules, used gateway nodes in a way to

prevent the blockage of possible connections and used a second interconnection round to improve connectivity. With these innovative aspects the algorithm achieved full connectivity in 97.28% of the 1250 test case scenarios, with up to 250 nodes. An emulation of the best state-of-the-art algorithm, meanwhile developed, even when added with our innovative aspects, only achieved 84.88% of fully connected scenarios.

1.4 Contributions

This thesis makes the following contributions:

- two communication topologies, named Group-Owner Client-Relay Group-Owner (GOCRGO) and Group-Owner Group-Owner (GOGO), which allow WFD inter-group communication for up to large scale scenarios. The former enables to form mesh like structures and the latter enables to form tree like structures. Both of them use the minimum number of devices and allow the use of unicast communication.
- a secondary communication topology named Group-Owner 2 Client-Relay (GO2CR) and a variant named Group-Owner Client-Relay UniCasts ($GOCR_{UC}$), which allow the use of unicast communication, but require more devices than the first two topologies.
- a network formation algorithm, called *RedMesh*, which defines five cluster interconnection rules, uses three communication rounds, and manage gateway nodes to increase connectivity.
- the identification of possible improvements in the *RedMesh* algorithm.

We also produced the following papers in the scope of this thesis:

- Core A, short paper: [49] A. Teófilo et al. "Group-to-Group Bidirectional Wi-Fi Direct Communication with Two Relay Nodes." In: *12th Int. Conf. on Mobile and Ubiquitous Systems: Computing, Networking and Services*. MobiQuitous'15. Coimbra, Portugal: ACM, July 2015, pp. 275–276. ISBN: 978-1-63190-072-3. DOI: 10.4108/eai.22-7-2015.2260272
- National conference, full paper: [50] A. Teófilo et al. "Comunicação Móvel Inter-Grupo Baseada em TCP sobre Wi-Fi Direct." In: *8th INForum - Simp. de Informática*. INFORUM'16. Lisboa, Portugal, 2016
- Core A, full paper: [51] A. Teófilo et al. "GOCRGO and GOGO: Two Minimal Communication Topologies for WiFi-Direct Multi-group Networking." In: *14th Int. Conf. on Mobile and Ubiquitous Systems: Computing, Networking and Services*. MobiQuitous'17. Melbourne, Australia: ACM, 2017, pp. 232–241. ISBN: 978-1-4503-5368-7. DOI: 10.1145/3144457.3144481

Therefore, with the communication topologies and the network formation algorithm created in this thesis, the ground is provided for systems that enable mobile resource sharing without the support of a communication infrastructure. Thus, the way is now paved to create autonomous mobile collaborative applications that can explore the ever increasing capabilities of smart phones.

This work can also be used in hybrid systems that have some parts that require device-to-device communication and other parts that may build bridges to a backbone, either by cable or wireless (Wi-Fi/3G/4G/5G). In [47] one case of such hybrid systems is presented, where, in simulation, data from mobile devices is shared by a Mobile Ad Hoc Network (MANET) network supported by the devices and shared by an infrastructure network where it is available. The work in this thesis could offer the background to bind the MANET used, in the mentioned system, to a real communication support based on the devices without the need to violate their warranty integrity.

1.5 Document structure

The reminder of this document is structured as follows.

Chapter 2 covers the research question *"Is it possible to do WFD intergroup connection and communication for large scale mesh networks?"*. Firstly, it presents the research context in WFD intergroup connection and communication, its current solutions and problems. Secondly, it contains our developed topologies for WFD intergroup connection and communication and also their evaluation.

Chapter 3 covers the research question *"Is it possible to systematically form WFD large scale mesh networks?"*. It starts presenting the current state-of-the-art algorithms for WFD network formation and identifies their major problems to support large scale mesh networks. Then it covers *RedMesh*, our developed WFD network formation algorithm and present its evaluation, its limitations and some reflections about improvements.

Finally, Chapter 4 concludes this dissertation and presents future work and open issues.

Chapter 2

WiFi-Direct Group Interconnection

This chapter covers the question *"Is it possible to do WFD intergroup connection and communication for large scale mesh networks?"*.

Section 2.1 introduces WFD intergroup connection and communication and existing challenges, and presents current solutions and their inefficiencies. Section 2.2 presents our newly proposed topologies, for WFD intergroup connection and communication, and their evaluation. Finally, Section 2.3 exposes our conclusions about the addressed research question.

2.1 Context and problems

Here we present the context and problems in WFD group interconnection and communication. We start with an introduction to WFD in Section 2.1.1 and then we discuss the problems to support WFD group interconnection and communication in Section 2.1.2. Next, we present current solutions and its limitations to achieve WFD group interconnection and communication in Section 2.1.3 and conclude the topic in Section 2.1.4.

2.1.1 WiFi-Direct introduction

Wi-Fi Direct (WFD) is a technology that enables direct device-to-device communication and is tailored to operate inside groups of devices. Its base services are device, and service, advertising and discovery, group authentication and formation, DHCP and data forwarding services.

Nodes start announcing their presence and trying to discover peer devices. They send probe requests in 802.11 channels 1, 6 and 11, in the 2.4 GHz band, and listen for probes of other devices. Each device elects one of these channels as its listen channel and only

listen on that channel. They alternate between sending and listening stages, spending on each state a randomly distributed amount of time between 100 and 300 ms.

After device discovery and before group formation, devices may announce their services and query for peers with specific services using the Generic Advertisement Service (GAS) (802.11u) [26]. Android devices may use the UPnP and the Bonjour [15] as high level discovery protocols to query for services. In this thesis as we only use the Bonjour protocol we call it: Bonjour Advertising Service (BAS). Devices also use this service to broadcast generic data to spread information to neighbour peers. With advertised information nodes discover their neighbour nodes and their state and they can proceed to group formation.

WFD groups have one node named Group Owner (GO) that acts as a group coordinator. This coordinator acts as an Access Point (AP) providing group authentication and formation, DHCP and data forwarding services to the remaining group members. Group formation can be done in three ways: *Standard Group Formation*, *Persistent Group Formation* and *Autonomous Group Formation*.

In *Standard Group Formation* two devices negotiate an *intent value* to be GO (a value in [0..15]) and the one with the highest value is elected. When the *intent values* are equal, devices use a random tie breaker bit (also sent) and the one with this bit active (generating new random bits if necessary) is the one elected. The elected node then defines the channel where the group will operate, creates an IEEE 802.11 Access Point (AP) and becomes ready to accept requests, from devices in the neighbourhood, in Wireless Protected Access - 2 (WPA2) [25]. The nodes then go through a Wi-Fi Protected Setup (WPS) [55] phase, establishing a secure connection and where the non-GO device connects by the WFD interface with the GO. Finally, the new client receives a DHCP address and is ready to use group services.

In *Persistent Group Formation*, devices use group credentials previously stored (from a past formed group with the involved devices). Hence, this case requires that, once a group is formed, devices save the group credentials and later, on a new connection, the devices will restore the GO and Client (CL) roles and re-stablish the saved group (between them). This possibility saves time, skipping the negotiating phase.

In *Autonomous Group Formation*, a device autonomously creates a group, becoming a GO on its own. Then, devices can discover this new GO and connect to it, also skipping the GO negotiation phase.

Once a group is formed, the GO can accept WFD clients, called CLs, but also WiFi clients, called Legacy Clients (LCs), if they know the Service Set Identifier (SSID) and Pre-Shared Key (PSK) of the announced network/group. Normally, GOs can only accept 8 clients. In generic terms we define that value as L and then $L = 8$. If a GO leaves the group, the group is broken and a new group has to be formed. There is no way of transferring the GO role to another node. All clients of a GO must be in the GO range and all the communication among them goes through the GO, being forwarded by the GO at MAC level. A detailed description of WFD can be found in [8].

Now we will analyse the sub-topics of: how to deal with Android WFD user connection approval and WFD characteristics to support multi-groups.

Android WFD user connection approval

Here we approach the practical detail of Android user connection approval. In a GO, the connection of a new client, for security reasons, requires a user approval action.

In this thesis as we would like to connect one device with several others and possibly changing connections over time, we do not want the inconvenience of asking the user for such confirmations. Therefore, to skip the user confirmation we set the GO device in WPS mode (push button method) to accept connections from other nodes (as clients). In this case the Android only shows a notification, but does not require a confirmation. That is not the correct way of doing it, as it weakens the security, but it was done for practical reasons.

This aspect calls for improvements in the Android interface, like having a way of disabling these notifications. Other possible ways are to present them as normal notifications or avoiding them by allowing connections by sending a PIN or the PSK to the GO and without having to activate the WPS push button method.

WFD characteristics to support multi-groups.

A WFD device can be connected to two WFD groups. It must be a LC in one group and it can be CL or GO in the other group. When a device is connected to two groups it must be active in only one of them at the time. Thus, when a device is inactive in a group, it should notify the group.

The WFD defines two power saving protocols that can be used to suspend the participation in a group. They are the Opportunistic Power Save Protocol (OPS) and the Notice of Absence Protocol (NoA).

The Opportunistic Power Save Protocol (OPS) allows clients to notify the GO that they will go into the inactive state. That allows them to be absent from the group, entering in a power save mode and be active in another group. This protocol also enables the GO to enter in an inactive state when all clients are in that same sate. Therefore, we conclude that this protocol can be used to allow the absence of clients by their own initiative.

The Notice of Absence Protocol (NoA) allows the GOs to notify clients of their absence from the channel (group), regardless of clients being in the active or in the inactive state. Those notifications establish NoA absence schedules using the parameters: delta time to first period, period duration, time between periods, and number of periods. A schedule can be cancelled or updated any time, with new notifications. Thus, we conclude that this protocol can be used to allow GOs absence by their own initiative.

We then conclude that WFD has the means to coordinate devices inside and among groups.

2.1.2 Group interconnection challenges

WFD specification does not tackle group interconnection, leaving this functionality open for research. Here we will see the challenges to connect two GOs directly and indirectly.

Two GOs can be connected directly if one connects its WiFi interface to the other. However, GOs usually have the same IP address (192.168.49.1/24) in the WFD interface and lower level communication layers eliminate packets from, or to, any address that is considered local. Therefore, two connected GOs will not be able to communicate directly using that address, as it is simultaneously a local and the destination address and that causes address conflict.

Two GOs may be connected indirectly by using a node as middle-point. That node should connect its WiFi and WFD interfaces to the GOs and is called a gateway or Client Relay (CR). Normally, it will not be able to address nodes from the groups of both GOs, as they all share the same network address 192.168.49.x/24. In these cases, the Android platform directs all packets to one interface, the *priority interface*. Consequently, relay nodes cannot normally initiate communications through their *non-priority interface*.

To make matters worse, the *priority interface* behaviour is not uniform. Some devices have WiFi as *priority interface* (e.g. Nexus 7), others have WFD (e.g. Nexus 5X, 6, 6P and 9) and others only support one active interface at a time (e.g. Motorola G2 2nd generation).

Also, Duan [13] and Casetti [10] reported that when devices have WiFi as their *priority interface*, broadcasts are sent out by both interfaces (WiFi and WFD). However, the opposite is not true, as we observed (in Nexus 6, Nexus 9 and OnePlus One) that when WFD is the *priority interface*, broadcasts are not sent by the WiFi interface.

With these difficulties, most of past WFD work focused on dealing with a single group or using a second communication technology or intermittent connections to cross WFD groups.

2.1.3 Group interconnection current solutions

To cross WFD groups we can use intermittent or permanent connections, as well as other technology. In Section 2.1.3.1 we discuss existing work when using another technology or intermittent connections. The use of permanent WFD connections is the most interesting solution as it offers WiFi speed and range, if devices can operate with both interfaces simultaneously and in an independent way. To the best of our knowledge, the only solution to offer WFD intergroup communication over persistent connections is from Duan et al [13], which is presented in Section 2.1.3.2.

2.1.3.1 Interconnection with no permanent connections

Inside one WFD group we identified the following works: Menegato et al [7] and Demir et al [11] proposed GO election schemes to form a group of WFD devices; Funai et al [18] proposed a task distribution system, using WFD to interconnect five devices in a WFD

group; Mao et al [33] studied data transmission strategies with three devices, inside one WFD group; Penner et al [39] formed a cloud with WFD group with three devices, to share tasks among devices; Pozza et al [42] used WFD to spread internet data through one WFD group, using several sharing schemes and comparing with BLT; and Rodrigues et al [44] benchmarked content dissemination in several wireless configurations, one of them was inside one WFD group.

One possible solution to cross WFD groups is to use a second technology. In these cases, WFD communication occurs just inside one group. Using Long Term Evolution (LTE) or Bluetooth (BLT) to communicate between WFD groups or with WFD groups and some internet service provider, we identified the following works. Asadi et al [2] connected three WFD groups with mobile LTE, they use groups with ten members but only in simulation; Pyattaev et al [43] analysed LTE traffic offloading onto WFD devices to share data inside WFD groups; Trestian et al [52] analysed energy consumption in multimedia data delivery comparing LTE, WFD and BLT, used one WFD group with one GO and one CL and their tests showed that WFD required less energy to transmit and receive data than BLT; and Gong et al [21] proposed a video streaming system, by enabling a WFD group of three devices to access a video server connected by LTE.

Some works use the WFD Generic Advertisement Service (GAS) to spread data in an augmented group of nodes. Marinho et al [35] implemented a routing messaging scheme over GAS, which became saturated with only 6 devices; and Shahin et al [46] proposed an alert dissemination protocol over GAS.

Nevertheless, the use of a second technology has the drawback of losing the WFD benefits of speed and range.

Besides that, Jung et al [30] proposed a self-organizing multi-group WFD network based on data location. However, they work in simulation and did not approach the limitations of WFD multi-group communication.

Delay Tolerant Networks (DTNs) [53] are networks where one device interconnects two WFD groups, but to transfer data, it connects to one of them at a time. DTNs enable intergroup communication, but require buffering and impose delays in data delivery. Example of works with WFD DTNs are: Hoang et al [23]; Felice et al [17]; Mizumura et al [36]; Funai et al [19, 20]; and Wong et al [57]. As Android allows multicasts to be sent by a specific interface, Funai et al [20] extended the DTN model, using multicasts to control the connections and the direction of the communications. Nevertheless, multicasts are almost as slow as broadcasts.

In summary, the address conflict and the priority interface limitations turn WFD intergroup communication in a non simple task. Presented solutions confirm that, as they use WFD but cannot make full use of WFD capabilities for intergroup communication.

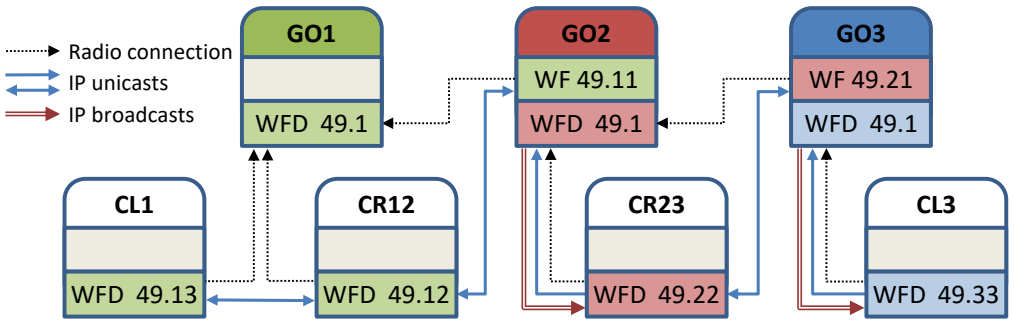


Figure 2.1: A 3 group scenario in GOCR topology.
GO nodes have identification and colour in top section. All nodes have, in their two lower sections, the IP and colour of the destination GO for WiFi and WFD interfaces, when connected. Similar scenarios use these features.

2.1.3.2 Group-Owner Client-Relay (GOCR) topology

The *Group-Owner Client-Relay (GOCR) topology*, proposed by Duan et al [13] and Casetti et al [10], enables direct GO connection. GO nodes connect their WiFi interface to the next GO, although they do not communicate directly. They use a client as an auxiliary node, which acts as a relay node (CR) to avoid the address conflict. To enable communication by the *non-priority interface*, GO nodes use broadcasts. This work assumes WiFi as the *priority interface*.

To provide a global perspective, Figure 2.1 presents a three group scenario using this topology. In this figure, and in similar ones, nodes contain identification and addresses in WiFi and WFD interfaces. To keep figures and text clear enough, addresses are only mentioned by their last two octets.

Figure 2.1 also shows interface radio connections (in black dash arrows), UDP unicasts (in blue solid arrows) and UDP broadcasts (in red double arrows). Communications will go over the established radio connections. In terms of message path and using GO2 to GO3 as an example, from GO2 to GO3, GO2 sends a broadcast to CR23 and this one relays the message in a unicast to GO3. CR23 can address GO3 because both are GO2 clients. The last message is addressed to 49.21 and packets are forwarded by GO2 at MAC level. In the opposite direction, GO3 sends a unicast to the CR23, which resends it in another unicast to GO2. GO3 can address CR23, as they are both GO2 clients and packets are sent by GO3 *priority interface* (the WiFi interface). Furthermore, communication will go from 49.21 to 49.22 without any conflict of address.

The relevant points in this topology are that GOs connect directly but communicate through one relay (CR) node and GOs must broadcast data to their CRs.

IP messages between CRs and the next group GOs are forwarded at MAC level by the GO of the CR, resulting in 2 MAC messages over the channel of the GO. We conclude that GOCR requires: 2 nodes per WFD group (1 GO and 1 CR) and per additional Wi-Fi Range (WFR)[1], and 3 MAC messages to cross a WFD group in each direction. It needs

[1] WFR is the conceptual communication range of WiFi and WFD.

broadcasts and also 3 routing operations (2 by GOs and 1 by CRs) in each direction.

This topology, to have all nodes interconnected, forms tree like networks, as each GO should connect its WiFi interface to another GO that is not its client or part of the networks of its clients. The tree top GO will be the only node with a free WiFi interface, which can be connected to any node. Nevertheless, it will not be enough to create a true mesh network.

2.1.4 Conclusion

This section showed that crossing WFD groups is not currently done in a way of potentially achieving WiFi speed and range. From all solutions, only GOCR topology goes in what we identify as the best direction: the simultaneous use of WiFi and WFD interfaces. Nevertheless, it presents many inefficiencies, namely requiring broadcasts, many nodes, data transmissions and routing operations per group and can only form tree networks. Therefore, it creates the opportunity and motivation to find topologies to use in WFD mesh networks that only communicate by unicasts.

Thus, we conclude that the question *"Is it possible to do WFD intergroup connection and communication for large scale mesh networks?"* is an open research question.

2.2 Solutions and results

Here we present the developed solutions and their results to address the question *"Is it possible to do WFD intergroup connection and communication for large scale mesh networks?"*.

In Section 2.2.1 we present our proposed topologies for WFD group interconnection. In Section 2.2.3 we evaluate existing and newly proposed topologies. Finally, in Section 2.2.4 we present our conclusions about WFD group interconnection.

2.2.1 Proposed group interconnection topologies

To create large scale networks it is necessary to have non-GO nodes that connect their WiFi and WFD interfaces to distinct groups. This solves the interconnections of LC[2] – GO and LC – LC, assuming that one LC will turn into a GO. The GOCR topology may complement these configurations providing interconnection for GO – GO.

Besides the LC – GO interconnection, we have the secondary goal of conceiving efficient topologies that use unicasts, a minimum number of nodes per WFD group and a minimum number of messages and routing operations to cross WFD groups.

With these goals, we present three new topologies that may rely only on unicasts: the *Group-Owner 2 Client-Relay (GO2CR) topology* [49], which needs two CRs between GOs for LC – GO interconnections; the *GOCRGO topology* [51] which needs just one CR between GOs for LC – GO interconnections and the *GOGO topology* [51] that allows direct

[2] Legacy Client (LC), a node that is client of a GO by connecting its WiFi interface.

Table 2.1: Communication assessment

	GO1 — $_{\text{WFD}}$CR$_{\text{WiFi}}$ — GO2				GO4$_{\text{WiFi}_{.41}}$ — GO5$_{\text{WiFi}_{.51}}$ — $_{.61}$WiFiGO6					
	GO1 ↔ CR		CR ↔ GO2		GO4 ↔ GO5		GO5 ↔ GO6		GO4 ↔ GO6	
	→	←	→	←	→	←	→	←	→	←
TCP	✓$_a$	✓$_a$	✗$_j$	✓$_c$	✗	✗	✗	✗	✗	✗
UDP	✓$_a$	✓$_a$	✗$_j$	✓$_d$	✗	.41$_f$	.61$_g$	.51$_g$	✗	✗
T/U-WF	–	✗	✓$_b$	–	.51$_e$	✗	.61$_h$	.51$_h$	.61$_i$	.41$_i$

T/U-WF: TCP and UDP sockets bound to WiFi interface

unicast communication between GOs in GO–GO interconnections. We also developed a variant of GOCR topology that only needs unicast communication.

All our developed topologies enable the exclusive use of TCP channels or UDP unicasts between nodes.

Now we will present the communication assessment that enables us to develop the topologies. After that we present the topologies.

2.2.1.1 Communication assessment

Android 5.0 (API 21) introduced the capability of binding TCP sockets to a specific interface, being the feature extended to datagram sockets in Android 5.1 (API 22). So, with Android 5 Compliant (A5C) devices it is possible to create sockets that force traffic out through a specific interface, circumventing the *priority interface*. Given this context, we will assess which communication possibilities are available to A5C and *non*-A5C devices using their WiFi and WFD interfaces. Table 2.1 lists the communication behaviour in two cases: two GOs interconnected by a relay node; and three GOs. Both cases were tested with Nexus 6, Nexus 9 and OnePlus One devices, all with WFD as *priority interface*. Table addresses just mention the last octet and they are mentioned only when they are not obvious. In the first line of the table, the addresses mentioned are the ones in the WiFi interface of each node. The addresses mentioned, in other lines, are the destination addresses used in the messages/datagrams. As UDP broadcasts and multicasts have slow communication speed, they are not used in the proposed topologies and are omitted from the table. Nevertheless, broadcasts can only be sent via WFD interface, while multicasts can be sent by both interfaces (if bound to the interface).

We begin by analysing the case of a CR in between GOs: case GO1–CR–GO2 in the table. On the CR *priority interface* side, both the CR and the GO1 can send UDP datagrams or create TCP sockets to the other (✓$_a$). On its *non-priority interface* side, the CR can communicate with the GO2 by sending UDP datagrams or creating TCP sockets if bound to the WiFi interface (✓$_b$). However, binding sockets to the WiFi interface is only available in A5C devices. If CR is a *non*-A5C one, it has to wait for a TCP connection coming from GO2 (✓$_c$) and leverage on the bi-directional nature of that connection. Although GO2 could send UDP datagrams to the CR (✓$_d$), the CR could not answer in the same protocol.

We move now to the analysis of GO to GO communication in case GO4–GO5–GO6 in Table 2.1. We start by assessing the situation of GO4–GO5, where GO4 is a client of GO5. As these GOs have the same IP address (192.168.49.1), they cannot communicate, between them, using that address. Ergo, to communicate, these devices must send the data addressed to the IP address, on the WiFi interface, of the other node: cases ($.41_f$) and ($.51_e$). The former ($.41_f$) is only possible because, unlike TCP, the UDP stack, at GO4, does not eliminate packets coming from an IP address that is also local (.1). The latter ($.51_e$) requires GO4 to create a unicast socket bound to its WiFi interface and direct all communication (TCP or UDP) to the address in WiFi interface of GO5. This enables: a) sockets to link to the correct interfaces, and to have a source address other than .1 (.41, in this case) needed for TCP; and b) to have a destination address also other than .1 (.51, in this case). The downside is that, to enable WiFi bounded sockets, GO4 must be an A5C device. In summary, data exchange between a GO_{CL} (GO4) that is a client of another GO_{GO} (GO5) is only possible if the GO_{CL} is an A5C device and the GO_{GO} has its WiFi interface connected. In that case, GO_{CL} can create a TCP connection to GO_{GO}, or both of them can use UDP datagrams.

Now we assess the GO5–GO6 connection. In this case GO5 and GO6 are both interconnected by their WiFi interfaces, and can communicate directly with UDP ($.61_g$, $.51_g$) or using UDP or TCP with sockets bound to WiFi ($.61_h$, $.51_h$), if they are A5C devices. All those sockets or datagrams must be directed to the IP address on the WiFi interface of the destination device. Lastly, GOs that are clients of one common GO, as GO4 and GO6, can communicate using UDP or TCP sockets bound to the WiFi interface and linked (addressed) to the address on the other GO WiFi interface ($.61_i$, $.41_i$).

In conclusion, it is possible to communicate, using UDP or TCP unicasts: (i) between GOs connected by a CR, if the CR is an A5C device or if it receives a TCP connection from its *non-priority interface*; and (ii) between GOs, if both are in the same network, if they have the WiFi interfaces connected, and, also, if the one that is client is an A5C device.

2.2.1.2 Group-Owner 2 Client-Relay (GO2CR) topology

In [49], we proposed the *Group-Owner 2 Client-Relay (GO2CR) topology*, which interconnects a pair of GOs using two CRs. For that purpose CRs must connect their WiFi and WFD interfaces, to the GOs, in a symmetrical way and each CR will only forward data from the *non-priority interface* to the *priority interface*. Consequently, CRs will send data only through their *priority interface*, and so they can depend only on unicast communication.

Figure 2.2 depicts the radio links (by WiFi and WFD interfaces) and communication paths, in a three WFD group scenario using this topology. likewise in [49], due to the characteristics of the devices used in tests, we consider WFD as the *priority interface* in Figure 2.2, here and in the remainder of this document.

Communication-wise, the GO2CR topology is characterized by the fact that a CR can

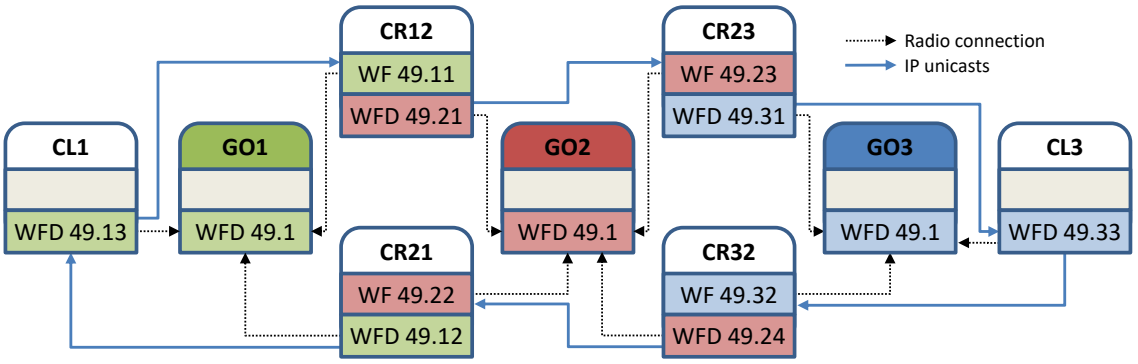


Figure 2.2: A 3 group scenario in GO2CR topology.

use its *priority interface* (WFD interface) to send unicasts directly, at IP level, to the next CR, being relayed at MAC level by the intermediary GO. Consequently this topology: needs an average of 1.5 nodes per WFR, as it needs 1 GO and 2 CRs per 2 additional WFRs; needs 2 hops (data transmissions and routing operations) to cross 1 WFD group; can use only *unicasts*; and enables to form mesh networks.

This topology can be used by devices with WiFi as the *priority interface*. It should be applied the same rule: CRs symmetrically forward data from the *non-priority interface* to the *priority interface*, which in this case is from the WFD interface to the WiFi interface.

2.2.1.3 Group-Owner Client-Relay Group-Owner (GOCRGO) topology

In [50, 51] we proposed the *Group-Owner Client-Relay Group-Owner (GOCRGO) topology*, that interconnects a pair of GOs using only one CR. The CR device will connect to the adjacent GOs using its WiFi and WFD interfaces.

In this topology, CRs have no problems creating TCP sockets or sending UDP datagrams from their *priority interface* side. But from the *non-priority interface* side, CRs cannot create any kind of socket without resorting to the features of Android 5. Yet, levering on the bidirectional nature of TCP connections, if a node from that side establishes a TCP connection to the CR, the CR can use that connection to send data to that side.

Figure 2.3 showcases the use of this topology, resorting only to the TCP protocol. It uses TCP connections from CL1 to $CR12_{49.11}$ (meaning address 49.11 in CR12), from CR12 to $CR23_{49.23}$, and from CR23 to $CL3_{49.33}$ or from CL3 to $CR23_{49.31}$, which will enable communication between CL1 and CL3. Communication from the CRs with the GOs is also possible, if for example each GO creates TCP connections to connected CRs. Furthermore, if CRs are A5C devices, it is also possible to communicate using only UDP datagrams.

To conclude, this topology needs an average of 1 node per WFR (1 GO and 1 CR per 2 WFRs) and needs 2 MAC messages and 2 routing operations to cross a WFD group. It can only resort to unicasts and can form mesh networks.

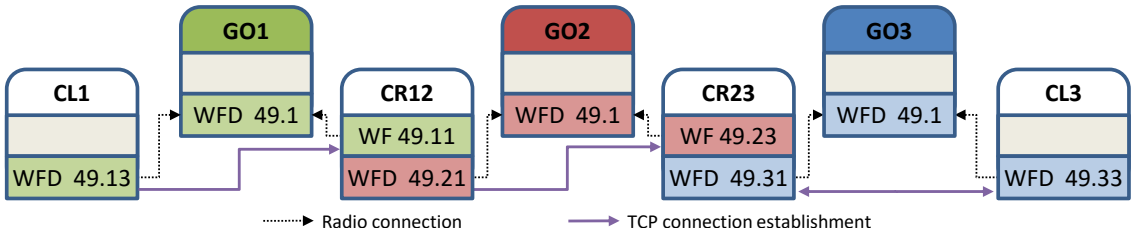


Figure 2.3: A 3 group scenario in GOCRGO topology.

This topology can be used directly by devices with WiFi as *priority interface*. With non-Android 5 Compliant (non-A5C) devices, each CR should receive a TCP connection from its *non-priority interface* interface. In Figure 2.3, considering WiFi as *priority interface*, we can create TCP connections from CL3 to $CR23_{49.31}$, from CR23 to $CR12_{49.21}$ and from CR12 to $CL1_{49.13}$ (or from CL1 to $CR12_{49.11}$).

2.2.1.4 Group-Owner Group-Owner (GOGO) topology

In [50, 51] we also proposed the *Group-Owner Group-Owner (GOGO) topology*, which enables direct GO–GO connection and communication. This topology has the goal of being more efficient than GOCR topology.

In this topology[3], GOs must have their WiFi interface connected to another GO. To communicate by TCP, a GO, connected by WiFi to another GO, can establish a channel (socket) to the other but it must go out by WiFi interface, requiring an A5C device. The channel must target the address in the WiFi interface on the other GO. This last requirement avoids the conflict of address and can be used because nodes decode that address as a local one. Having a TCP channel created, which is bidirectional, both nodes can exchange messages. To communicate by UDP, a GO (GO1), connected by WiFi to another GO (GO2), the first one (GO1), if it is an A5C device, can send unicasts from the WiFi interface (*non-priority interface*) to the second (GO2), if they are addressed to the WiFi address of the second (GO2). In the same scenario, GO2, if it is an A5C device, can send UDP unicasts to GO1 if sent from the WFD interface (*priority interface*) and targeted to WiFi address of GO1. These UDP unicasts are not suppressed by the Android/Linux communication layers. GO1 like nodes, must be A5C devices to allow UDP unicasts to be sent through the *non-priority interface*.

Figure 2.4 depicts the use of this topology in a 3 group scenario. It can be seen that all GOs have the WiFi interface connected, offering an alternative IP address, other than 49.1, to avoid conflicts of address. For data communications we use the notation "$D_i \rightsquigarrow E_a$"[4] to denote a data transmission from device D, bound to interface i, to address a of device E. If i is omitted, it means that transmission uses *priority interface* or the only connected interface. As now expected, transmissions by *non-priority interface* must be *bound* to that interface.

[3] By default, we consider WFD as *priority interface*.
[4] To make it simple, addresses only mention the last octet.

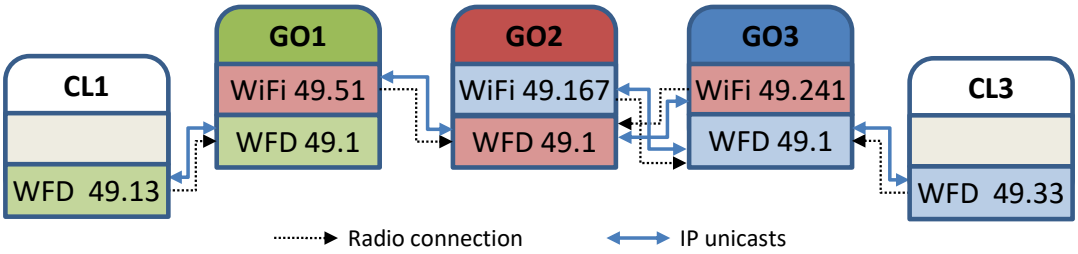


Figure 2.4: A 3 group scenario in GOGO topology.

In that scenario, a UDP communication from CL1 to CL3 follows path CL1 $\rightsquigarrow$ GO1$_{.1}$, GO1$_{\text{WiFi}}$ $\rightsquigarrow$ GO2$_{.167}$, GO2 $\rightsquigarrow$ GO3$_{.241}$, and finally GO3 $\rightsquigarrow$ CL3$_{.33}$[5]. In turn, a communication from CL3 to CL1 follows the path CL3 $\rightsquigarrow$ GO3$_{.1}$, GO3 $\rightsquigarrow$ GO2$_{.167}$, GO2 $\rightsquigarrow$ GO1$_{.51}$, and finally GO1 $\rightsquigarrow$ CL1$_{.13}$[6].

To use TCP, between CL1 and CL3, we can establish the following bidirectional connections: CL1 $\rightsquigarrow$ GO1$_{.1}$, GO1$_{\text{WiFi}}$ $\rightsquigarrow$ GO2$_{.167}$, GO2$_{\text{WiFi}}$ $\rightsquigarrow$ GO3$_{.241}$ and finally GO3 $\rightsquigarrow$ CL3$_{.33}$.

Consequently, this topology needs 1 node per WFR, and needs 1 MAC message and 1 routing operation to cross a WFD group. It can also only resort to unicasts and it forms tree like networks (like GOCR).

If GOs have WiFi as the *priority interface*, and even if devices are non-A5C, each GO can create TCP sockets from its WiFi interface to the WiFi address of next GO, allowing bi-directional communication between them. In Figure 2.4 we can create the following TCP sockets (or paths to UDP datagrams): GO1 $\rightsquigarrow$ GO2$_{.167}$; GO2 $\rightsquigarrow$ GO3$_{.241}$; and GO3 $\rightsquigarrow$ GO2$_{.167}$. Furthermore, GO2 can create UDP sockets to GO1$_{.51}$, but only if bound to the WFD interface (A5C property). Therefore, this topology, with WiFi as the *priority interface*, can use TCP connections with no restrictions, but needs A5C devices to allow UDP communication.

The GOGO topology is better than GOCR in every aspect, except that it requires A5C GOs while GOCR does not impose any restrictions. We present a detailed comparative analysis between all four topologies (GOGO, GOCR, GOCRGO and GO2CR) in the immediate section (2.2.2).

2.2.2 Topologies analysis

In this section we perform a comparative analysis of the proposed topologies (GO2CR, GOCRGO and GOGO) and the current state-of-the-art topology (GOCR) concerning their spatial node requirements, communication speed, routing demands, frequency usage, network redundancy, connection flexibility and behaviour in extreme situations.

[5] Path can be shortened in 1 hop by either: GO1$_{\text{WiFi}}$ $\rightsquigarrow$ GO3$_{.241}$; or GO2$_{\text{WiFi}}$ $\rightsquigarrow$ CL3$_{.33}$.
[6] Path can be shortened in 1 hop by either: CL3 $\rightsquigarrow$ GO2$_{.167}$; or GO3$_{\text{WiFi}}$ $\rightsquigarrow$ GO1$_{.51}$.

Table 2.2: Topologies spatial node requirements.

Topology	#GOs	#CRs	Range (in WFRs)	#nodes/WFR
GOCR	1	1	1	2
GO2CR	1	2	2	1.5
GOCRGO	1	1	2	1
GOGO	1	0	1	1

#GOs and #CRs columns contain the nodes per additional group, which extends the range by the value in the Range column.

Spatial node requirements

We begin by analysing the node density (the number of nodes) imposed by each topology. For that purpose, we consider the conceptual measure Wi-Fi Range (WFR), which denotes the maximum coverage distance of a WiFi (and WFD) radio for a mobile device.

GOCR topology requires 1 GO and 1 CR per additional WFD group, which extends the coverage range by WFR, giving 2 nodes per WFR. GO2CR topology requires 1 GO and 2 CRs per additional group, extending the coverage range by 2 WFRs, giving an average of 1.5 nodes per WFR. GOCRGO topology requires 1 GO and 1 CR per additional group, extending 1 WFR, giving an average of 1 node per WFR. Lastly, GOGO topology requires 1 GO per additional group, extending 1 WFR, giving 1 node per WFR. This information is summarized in the Table 2.2.

We, then, conclude that the GOGO and GOCRGO topologies present the best average values for #nodes/WFR. Among them, GOGO provides the best radio coverage as it builds only on GO nodes.

Communication speed

We now want to assess the maximum communication speed in one direction, while in the presence of bi-directional simultaneous communications. For our analysis we make three assumptions: i) GOs operate on independent WiFi channels without any interference; ii) devices can communicate simultaneously on both interfaces; and iii) for sake of simplicity, we make the rough approximation of considering the same speed for TCP and UDP communications. We consider the following conceptual measures/terms:

Wi-Fi Speed (WFS) – the unicast maximum communication speed in WiFi (and WFD);

Broadcast Speed (BCS) – the maximum broadcast speed;

Broadcast Factor (BCF) – the factor between the maximum unicast and the broadcast speed ($BCF = WFS/BCS$);

Smax – the maximum communication speed in one direction, when the topology is used in a single direction; and

Table 2.3: Topologies communication speed.

Topology	SMAX ($Mbps^*$)	SBDMAX ($Mbps^*$)
GOCR	WFS/(2 + BCF) = 4.9, WFS/3 = 18.0	WFS/(5 + BCF) = 3.86
GO2CR	WFS/2 = 27.0	WFS/4 = 13.5
GOCRGO	WFS/2 = 27.0	WFS/4 = 13.5
GOGO	WFS/1 = 54.0	WFS/2 = 27.0

SBDMAX – the maximum communication speed in one direction, when the topology is simultaneously used in both directions.

For example, between two directly connected devices, UDP unicast communication has SMAX = WFS and SBDMAX = WFS/2, since, in the latter, the channel has to be shared between two unicast communications.

To give a sense of proportion, we consider the concrete values of $WFS = 54 Mbps$ and $BCS = 6 Mbps$, leading to $BCF = 9$, that is, one broadcast will be equivalent to 9 unicast messages over the channel. Every result computed from these assumptions is marked with *.

For short, we will use term "message" to denote either UDP unicast datagrams or data over TCP channels. Additionally, both IP and MAC messages will be, by default, unicast messages. On the other hand, UDP broadcasts will be mentioned as just broadcasts. The results of our analysis are summarized in Table 2.3.

In GOCR, and considering Figure 2.1, from right to left, a GO has to handle 1 IP message from the GO at its right to its CR (2 MAC messages), and 1 IP message from the CR to the GO itself. That sums up 3 MAC messages in the channel of the GO, and so SMAX = $WFS/3 = 18 Mbps^*$. From left to right, a GO has to send a broadcast to its CR, which must forward the message as an IP message to the GO at its right, taking 2 MAC messages in the channel of the GO. This sums up $2 + BCF$ MAC messages, thus SMAX = $WFS/(2 + BCF) = 4.9\ Mbps^*$. Consequently, each GO channel has to support $3 + 2 + BCF$ messages, resulting in SBDMAX = $WFS/(5 + BCF) = 3.86\ Mbps^*$.

In GO2CR, communication is symmetric, and in each direction requires 1 IP message between CRs, resulting in 2 MAC messages over the channel, hence SMAX = $WFS/2 = 27\ Mbps^*$, leading to SBDMAX = $WFS/4 = 13.5\ Mbps^*$.

In GOCRGO the behaviour is similar to GO2CR, 1 IP message between CRs, so the results are the same.

Finally, in GOGO every GO channel carries a single IP message in either direction. Ergo, SMAX = WFS = $54\ Mbps^*$ and SBDMAX = WFS/2 = $27\ Mbps^*$.

From these values, we conclude that GOGO is the fastest topology.

Table 2.4: Topologies routing operations.

Topology	# IPRO	# MACRO	# RO	# RO/W
GOCR	1 GO + 1 CR	1 GO	3	3 / 1 = 3
GO2CR	1 CR	1 GO	2	2 / 2 = 1
GOCRGO	1 CR	1 GO	2	2 / 2 = 1
GOGO	1 GO	0	1	1 / 1 = 1

Routing operations

Here we discuss the routing operations needed to cross WFD groups, for each of the four topologies under analysis. Here, we want to see how many MAC and IP routing operations are needed to cross a group. For sake of readability, we will use IPRO and MACRO to designate a routing operation at IP and MAC level, respectively.

The GOCR is a symmetric topology with respect to routing. Considering Figure 2.1 once again, starting at GO2, this device performs an IPRO and sends a broadcast to CR23, which, in turn, performs another IPRO and sends a message to GO3 requiring one MACRO by GO2. In the opposite direction, the routing behaviour is symmetrical. Hence, in each direction, communication requires 3 routing operations (2 IPROs and 1 MACRO) to cross one group.

The GO2CR topology is symmetric with relation to routing, but each CR routes traffic in a single direction. This topology requires 1 GO MACRO and 1 CR IPRO, in a total of 2 routing operations, in each direction.

The GOCRGO behaves similarly to GO2CR, but CRs route all the traffic in the channel. Therefore it needs 2 routing operations.

The GOGO communicates directly between GOs, so it requires a single GO IPRO to cross a group (1 routing operation).

Finally, given that the topologies present distinct spatial behaviours, to ensure fairness, our analysis must also take into consideration the number of routing operations per WFR, we designate it RO/W and use it in terms $\text{TOPOLOGY}_{\text{RO/W}}$. In both GOCR and GOGO topologies, 1 WFD group can extend the network by 1 WFR, hence $\text{GOCR}_{\text{RO/W}} = 3/1 = 3$ and $\text{GOGO}_{\text{RO/W}} = 1/1 = 1$. In the case of both GO2CR and GOCRGO topologies, 1 WFD group can extend the network by 2 WFRs, so $\text{GO2CR}_{\text{RO/W}} = \text{GOCRGO}_{\text{RO/W}} = 2/2 = 1$. This information is summarized in Table 2.4, where the first three columns with values contain the number of routing operations (IP, MAC and total) per WFD group.

Thus, we conclude that all proposed topologies (GO2CR, GOCRGO and GOGO) perform optimally in terms of number of routing operations per distance, as they require 1 operation per WFR. However, in crowded scenarios, as we will seen in Extreme Situations analysis, when GOs are near each other, and CRs will not make any useful extension of range, the number of routing operations per group is more relevant. In those cases, GOGO should be the preferred topology, as it only offers one routing operation to cross groups.

Network frequency usage

Here we want to know the number of non-overlapping frequencies needed, by the topologies, to offer interference-free communication.

Concerning frequency, or radio channel, analysis, GO2CR and GOCRGO are identical, as they need intermediary nodes (1 or 2) to connect GOs. Likewise, GOCR and GOGO topologies are also identical, as both require direct GO–GO connection. Thus, for sake of simplicity, we confine our analysis to GOCRGO and GOGO, here and in the remaining analyses.

Therefore, to support our comparison and analysis, Figures 2.5 and 2.6 illustrate the application of those topologies to the same bi-dimensional[7] (2D) scenario. These figures use $L = 8$, meaning that GOs support eight clients. They depict the radio connections (WiFi or WFD) between CLs/CRs and GOs, always directed from the client to the GO, and display the channels in use by the GOs with colours and letters (in their own nodes). Figure 2.6 also shows the direction of the WiFi interface connections in GO–GO connections. We should note that GO4 and GO5 are double connected, because, in GOGO, all GOs must have their WiFi interface connected.

The GOCRGO, in conflict-free radio communication and considering GOs in straight line (1D), needs only two non-overlapping frequencies. Nevertheless, as CRs should communicate without interferences, it requires that consecutive GOs with the same frequency should be more than three WFRs apart. This can be observed in the GO1–GO3 row of nodes in Figure 2.5. Regarding bi-dimensional scenarios, we empirically claim that GOCRGO needs at least six non-overlapping frequencies to avoid interference. For example, Figure 2.5 with three frequencies may have interferences between the southwest CL of GO2 and one possible CL of GO5 positioned near the northeast coverage area of GO5. In Figure 2.7a we can see that if we have nodes positioned in a grid, with cell size of WFR, the scenario does not have interferences if six channel frequencies are used. We can see that, in the scenario, GOs are more than three WFRs apart from other GOs using the same frequency.

The GOGO, in straight line conflict-free communication between GOs, requires four non-overlapping frequencies. GOs using the same frequency, should have three GOs between them and be at more than three WFRs apart. Frequencies should be assigned in a circular way, like A, B, C, D, A, B, ..., as can be observed in the GO4–GO8 line of nodes in Figure 2.6. To expose why four frequencies are needed, lets consider those GO4–GO8 nodes. From the perspective of GO6: GO5 and GO7 should have distinct frequencies and GO8 should not use frequency of GO6 in order not to cause interference in GO7. Therefore, a frequency can only be used once every four GO nodes in line. Regarding bi-dimensional scenarios, we empirically claim that GOGO needs at least ten non-overlapping frequencies to avoid interference. For example, Figure 2.6 with six frequencies may have interferences between the southwest CL of GO2 and one possible

[7] Relative to node disposition.

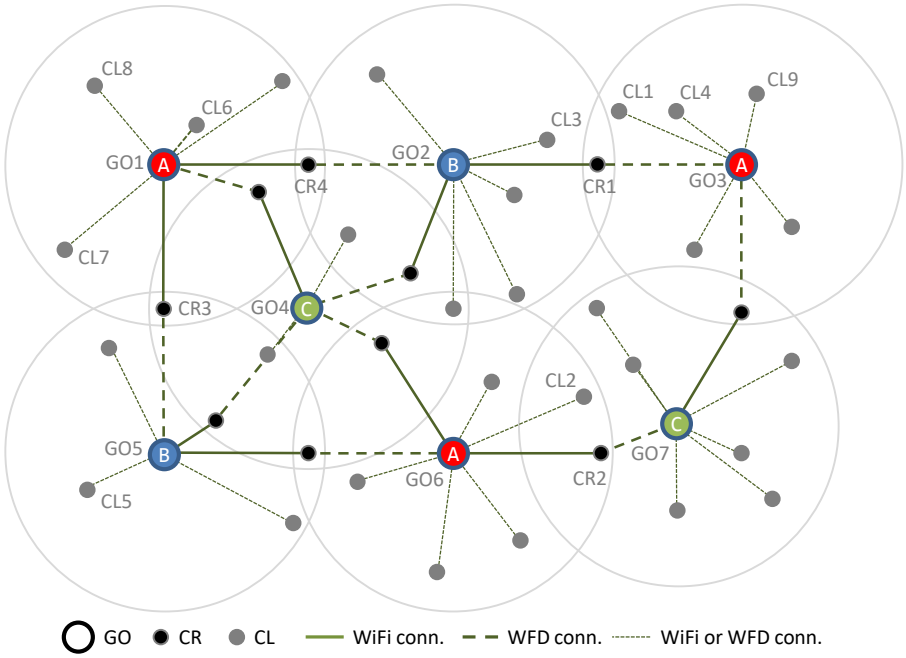


Figure 2.5: GOCRGO topology applied to a 2D scenario.

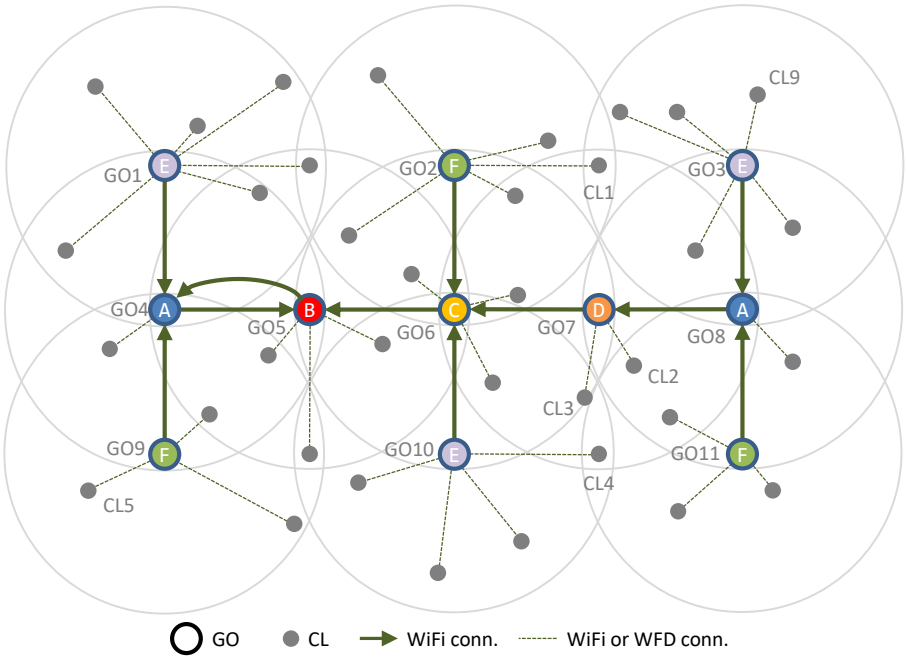
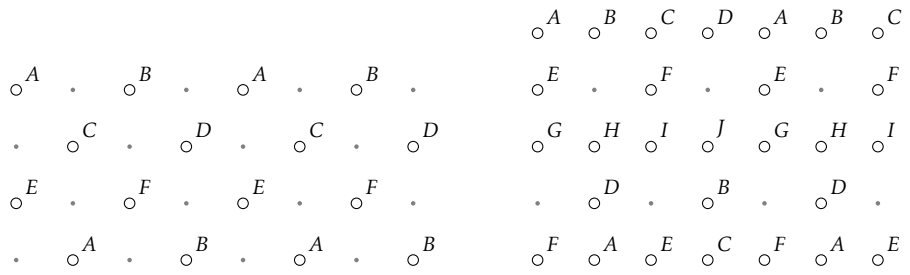


Figure 2.6: GOGO topology applied to a 2D scenario.

CL of GO9 positioned near the northeast coverage area of GO9. In Figure 2.7b we can see that if we have nodes positioned in a grid, with cell size of WFR, the scenario does not have interferences if ten frequencies are used. We can also see that in this scenario GOs are more than three WFRs apart from other GOs using the same frequency.

Thus, we conclude that GOCRGO and GO2CR topologies cover more area per WiFi channel and need fewer non-overlapping frequency channels to communicate without



a: GOCRGO, WFR = $\overline{AB}$ / 2. b: GOGO, WFR = $\overline{AB}$.

Figure 2.7: GOCRGO and GOGO in a 2D grid scenario.
Small grey circles are available nodes (that can be CRs, CLs or LCs) and big white circles are GOs nodes. Letters mean channel frequencies.

collisions, than GOGO and GOCR.

Network path redundancy

Here we analyse the potential to create networks with redundant paths, for the GOGO and GOCRGO topologies. They are representative of GOCR and GO2CR topologies, respectively. We start by characterizing the network structure of the topologies and, subsequently, analyse their potential to create redundant paths.

Network structure. The GOCRGO topology can be used to connect GOs and CRs according to a mesh structure, like in Figure 2.5. In turn, GOGO forcibly connects GOs in a tree structure, due to the need to connect the WiFi interface of every GO to another GO (not in the client network of the first one). The tree top node is the only exception and can be used to connect the WiFi interface forming a loop, like the connections between nodes GO4 and GO5 in Figure 2.6.

Path redundancy. The mesh structure of GOCRGO may originate *multiple paths* between distant nodes (per example, paths: GO1–GO2–GO3–GO7 and GO1–GO5–GO6–GO7, in Figure 2.5), as well as between adjacent GOs (per example, paths: GO2–GO4 and GO2–GO1–GO4 also in Figure 2.5). Thus, GOCRGO may have path redundancy at any scale.

Conversely, the tree-based structure of GOGO cannot build redundant paths. The only exception are the ones that use the loop that connects the top of the tree to another GO (GO4–GO5, in Figure 2.6).

Thus, in GOCRGO, we may leverage on the multiple redundant paths to perform network *traffic splitting*. For instance, given that in GOCRGO the maximum communication speed per link (Smax) is $WFS/2$, n alternative paths between two areas may increase such speed to $n \times WFS/2$.

In turn, in GOGO, the top tree GO has to support all the traffic between the GOs connected to it and their trees of connected nodes, which will cause a major bottleneck.

Redundant paths enable the choice of *shorter paths* and that improves the *energy efficiency* of the network. Redundant paths also enable the choice of paths with better resources, like more battery, or less load, and that may prevent node energy exhaustion and defer network reconfigurations as much as possible. That leads to better *energy management* and less *network maintenance* costs.

Path redundancy in GOGO is confined to the previously mentioned top loop, but that alone should not be enough to avoid long and congested paths. In Figure 2.6 we can observe that traffic between clients of GO2 and GO3 must follow a long path (GO2, GO6, GO7, GO8 and GO3), albeit the proximity of these nodes, and, as already mentioned, traffic between nodes at opposite edges of the network must always travel across GO4, GO5, GO6, GO7 and GO8, contributing to a communication bottleneck.

In summary, one can leverage on the path redundancy provided by GOCRGO to implement a smart routing strategy that makes use of traffic splitting to improve communication speed and reduce energy consumption and network reconfiguration (due to the lack of battery power).

Network flexibility

Here we assess how topologies react to node churn, which is the *egress* and *ingress* of nodes.

In GOCRGO, path redundancy provides general resiliency to node churn and in particular to the *egress* of CRs. For instance, in Figure 2.5, if we remove a single CR, any other node will remain connected. Even if both CR1 and CR2 leave the network, connectivity may be re-established by replacing CR2 by CL2, or by promoting CL1 to the role of Group Owner to interconnect GO2 and GO3, using CL3 and CL4 as CRs. The *egress* of a GO will require some reconfiguration to reconnect its clients, but the operation should be mostly local if there are nodes available nearby. Broader reconfigurations may occur when node distribution is very sparse. Usually, the exit of a GO can be solved by electing a nearby CL to become a GO.[8] However, if there are still unconnected nodes, another GO, or GOs, must be elected to connect them. For example, in Figure 2.5, if GO1 runs out of battery, CL6 can become a new GO (GOx6). However, as GOx6 cannot reach either CL7 or CR3, CL7 may also become a GO (GOx7) and connect to CR3. In such reconfiguration, with the exception of CR3 and CL7, all the former clients of GO1 should connect to GOx6.

In GOGO, the *egress* of GOs may require significant and non-local changes because the tree-like structure must be preserved, with the GOs WiFi connections directed to the root of the tree. Looking once again at Figure 2.6, if GO7 fails, we can promote CL2 and CL3 to GOs to create a path between GO6 and GO8. However, if for some reason those nodes are not available, we will have to resort to CL1 or CL4. Promoting CL1 to a GO (GOx1) that interconnects GO2 and GO3, forces GOx1 to connect its WiFi interface to

[8] The electing criterion is out of the scope of this thesis.

GO2, requiring GO3 to disconnect its WiFi interface and connect it to GOx1, and GO8 to disconnect and connect to GO3. A similar situation arises if we promote CL4 to a GO, to interconnect GO11 and GO10. Admittedly, the extent of connection reversal depends on the place in the tree where the reconfiguration must be performed. So, unlike GOCRGO, *node egress* in GOGO may require non-local reconfigurations even when node distribution is dense.

Regarding *node ingress*, there are many scenarios to consider. Nonetheless, in all the scenarios we analysed, GOGO always required more actions than GOCRGO. We showcase how the topologies can react to a new node with two paradigmatic scenarios.

Scenario 1: a new node, Xa, comes in reach of (and only of) CL5, at the bottom left in both Figure 2.5 and Figure 2.6. In GOCRGO: 1) Xa becomes a GO (GOa); and 2) CL5 connects its available interface to GOa. In GOGO: 1) CL5 disconnects from GO9; 2) CL5 switches into a GO (GOb); 3) GOb connects via WiFi interface to GO9; and 4) Xa connects to GOb.

Scenario 2: two non-connected networks, Na and Nb (replicas of Figure 2.5 or of Figure 2.6), get in range of each other, with Na located at bottom left and Nb at the top right, and only nodes CL9 from Na ($CL9_{Na}$) and CL5 from Nb ($CL5_{Nb}$) are in range of each other. In GOCRGO (both Na and Nb are replicas of Figure 2.5): 1) $CL9_{Na}$ disconnects from $GO3_{Na}$; 2) $CL9_{Na}$ becomes a new GO (GOx); 3) $CL4_{Na}$ connects with its free interface to GOx; and 4) $CL5_{Nb}$ connects to GOx. In GOGO (both Na and Nb are replicas of Figure 2.6), both $GO3_{Na}$ and $GO9_{Nb}$ have their WiFi interfaces connected to other nodes, thus the networks cannot be interconnected without reconfiguring the structure of at least one of them. Hence, a runtime analysis is necessary to choose which network will require fewer structural changes to be connected to the tree of the other network, forming a single global tree. If we choose to connect Nb to Na, the following actions are necessary: 1) $CL5_{Nb}$ disconnects from $GO9_{Nb}$; 2) $CL5_{Nb}$ turns into a GO ($GOx5_{Nb}$); 3) $GO4_{Nb}$ disconnects from $GO5_{Nb}$; 4) $GO4_{Nb}$ connects to $GO9_{Nb}$; 5) $GO9_{Nb}$ disconnects from $GO4_{Nb}$; 6) $GO9_{Nb}$ connects to $GOx5_{Nb}$; 7) $CL9_{Na}$ disconnects from $GO3_{Na}$; 8) $CL9_{Na}$ turns into a GO ($GOx9_{Na}$); 9) $GOx9_{Na}$ connects to $GO3_{Na}$; and lastly, 10) $GOx5_{Nb}$ connects to $GOx9_{Na}$. Alternatively, connecting Na to Nb requires changing the connections between nodes GO5, GO6, GO7, GO8 and GO3 of Na.

In summary, the GOCRGO mesh structure, with redundant paths, can tolerate some structural network changes without requiring compensation actions, and when some adjustment actions are required they should be mostly local. In GOGO, structural network changes will often require reconfigurations that reach far beyond the place where the change occurred. In the light of these findings, we conclude that GOCRGO is more flexible than GOGO.

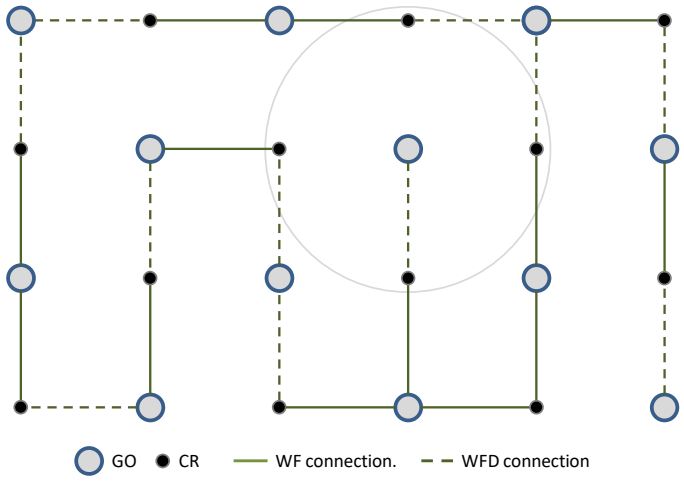


Figure 2.8: Application of GOCRGO topology to a sparse scenario.

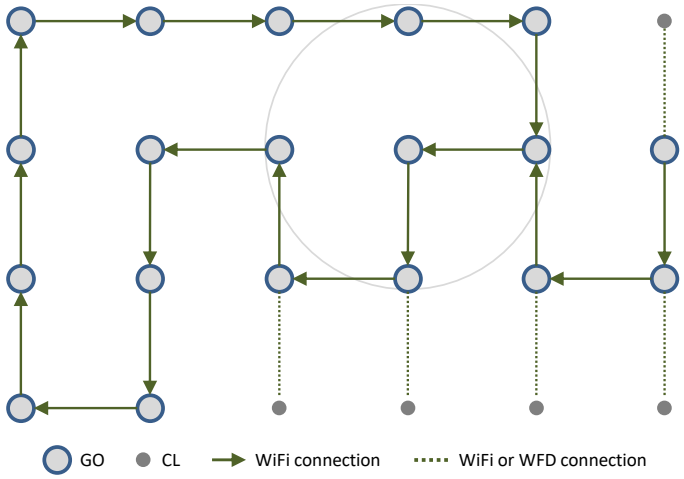


Figure 2.9: Application of GOGO topology to a sparse scenario.

Extreme situations: sparse and crowded scenarios

Here we discuss the behaviour of the topologies in the extreme cases of sparse and crowded scenarios.

In *sparse scenarios*, nodes are at the limit of the radio range. Figures 2.8 and 2.9 illustrate the application of the GOCRGO and GOGO topologies, respectively, in the same sparse scenario. Both figures show the radio range of one node and the nodes disposed in a grid of equally sized square cells. In this scenario, nodes are so sparse that it is very difficult to create redundant paths with GOCRGO. Thus, GOCRGO has SMAX = $WFS/2$ and GOGO has SMAX = WFS. Also, as every node in GOGO can be a GO, GOGO provides better radio coverage than GOCRGO.

The fact that the nodes are at the coverage boundaries precludes the use of the GO2CR topology, because there are not enough nodes to interconnect two adjacent GOs, over all the scenario. However, GOCR topology can still be used in this particular scenario, because it follows a $4 \times n$ node distribution. In those cases, both middle lines should

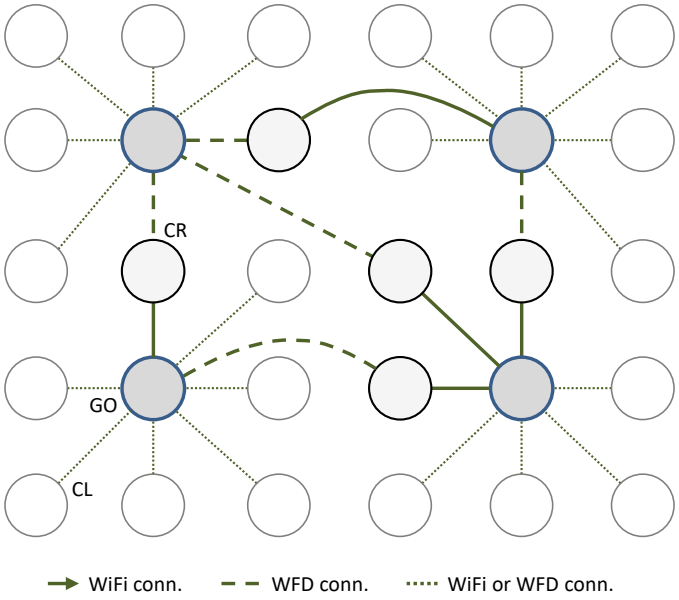


Figure 2.10: A crowded scenario with GOCRGO topology.

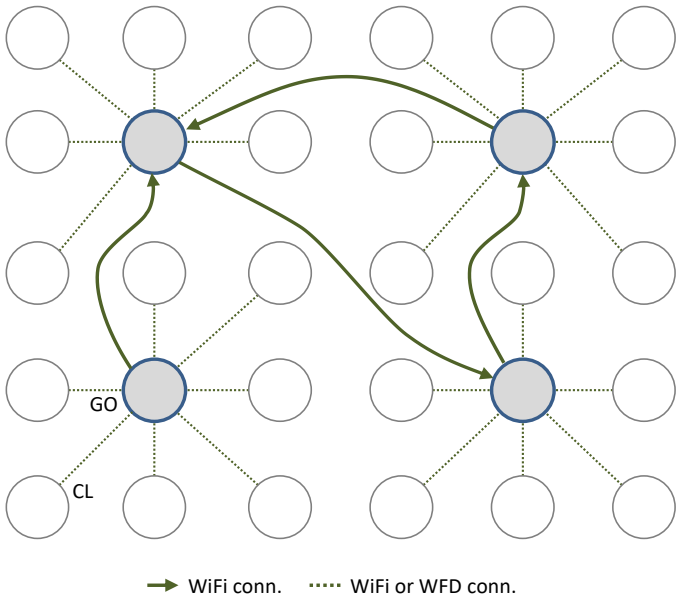


Figure 2.11: A crowded scenario with GOGO topology.

be composed only by GOs and the other lines composed by CLs, where each CL should connect to the nearest GO and act as its relay node. In the top/bottom line of GOs, each GO should connect its WiFi interface to the GO on its left/right, respectively. Finally the top left GO should connect its WiFi interface to the bottom left GO.

In *crowded scenarios*, like in a stadium, nodes are very close to each other. Figure 2.10 and Figure 2.11 depict, respectively, the application of the GOCRGO and GOGO topologies in the same crowded scenario. In this kind of scenario, all the nodes are in radio coverage range of all the others, so any node can connect to any other. With nodes so close, every simultaneous transmission in the same or in neighbour WiFi channels will

Table 2.5: Topologies comparison

Topology	Nodes per WFR	Comm. Speed SBDMAX	Routing per WFR	Freqs per 2 WFRs	Freqs needed 1D/2D	RC	RP TS SP BEM ECS NF	ES S/C	AD
GOCR	2	WFS/14*	3	2	4 / 10†	+	✗	✗/–	ANY
GOGO	1	WFS/2	1	2	4 / 10†	+	✗	+/+	A5C
GO2CR	1.5	WFS/4	1	1	2 / 6†	–	✓	✗/–	ANY
GOCRGO	1	WFS/4	1	1	2 / 6†	–	✓	–/–	ANY

* SBDMAX = $WFS/(5+BCF)$, and using $BCF = 9$ ($WFS = 54$, $BCS = 6$)
† Empirical value.
Acronyms: RC, Radio coverage; RP, Redundant Paths; TS, Traffic Splitting; SP, Short Paths; BEM, Better Energy Management and Efficiency; ECS, Extended Communication Speed; NF, Network Flexibility; ES, Extreme Situations; S/C, Sparse / Crowded scenarios; AD, Android Device; A5C = Android 5 Compliant device.

cause radio interferences. Consequently, in GOCRGO, redundancy with short paths and traffic splitting cannot be helpful, because every communication will probably overlap with many others. Also, when in the presence of high interference, each communication should be as close as possible, otherwise success will be very limited. In turn, GOGO will use less bandwidth than GOCRGO because it only requires 1 message to cross WFD groups, against the 2 messages required by GOCRGO. Furthermore, GOGO also only needs one routing action to cross WFD groups, against the two required by GOCRGO. It is thus expected that GOGO will be faster and more energy efficient than GOCRGO in these kind of scenarios.

From the above we conclude that GOGO can offer better communication speed in sparse and crowded scenarios.

Final analysis

Table 2.5 summarizes our findings. Given that GOGO is similar to GOCR, as they both use direct GO to GO connection, and GOGO is always better than or equal to GOCR, we choose GOGO as the best option from both. The only exception is that GOGO needs A5C devices, but nowadays that should not be a problem. The same happens with GOCRGO and GO2CR, as both need relay nodes in-between GOs and as the former is always better than or equal to the latter, GOCRGO is the best option.

Comparing GOGO and GOCRGO, the *strong points of GOGO*, in relation to the other, are communication speed, radio coverage and behaviour in extreme cases. While the *strong points of GOCRGO*, in relation to the other, are network path redundancy, traffic splitting, shorter paths, better energy management and efficiency, extended communication speed, better network flexibility, better frequency usage and the ability to be used by any Android device when only using TCP connections. Taking into consideration these

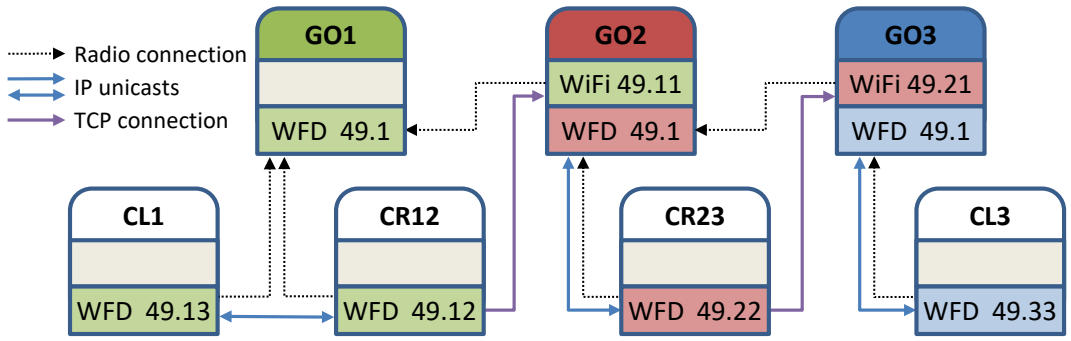


Figure 2.12: A 3 group scenario in GOCR$_{UC}$ variant.

results, our conclusion is to choose GOGO for extreme scenarios and GOCRGO for all the other cases. Furthermore, both topologies can co-exist in networks with a wide diversity of node concentration, being GOGO suitable for building tree networks and GOCRGO for mesh networks.

2.2.3 Topologies evaluation

We evaluate the performance of all topologies (GOCR, GOGO, GO2CR and GOCRGO) regarding communication speed and energy consumption.

The experimental testbed is composed of Nexus 6 and Nexus 9 phones, all with WFD as *priority interface*.

Given that GOCR was proposed for devices with WiFi as *priority interface*, we develop a variant, which we named Group-Owner Client-Relay UniCasts (GOCR$_{UC}$) [51] to allow the use of that topology in our devices.

Therefore, before presenting the evaluation, we first present the GOCR$_{UC}$ variant.

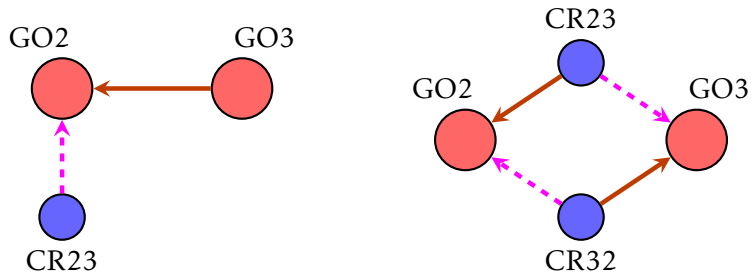
Group-Owner Client-Relay UniCasts (GOCR$_{UC}$) variant

GOCR$_{UC}$ is, thus, a variant of GOCR that has WFD as *priority interface*. To respect the nature of GOCR, GOCR$_{UC}$ keeps its radio connections and does not use Android 5 capabilities. In this setting, since GOs cannot send unicast traffic through their WiFi interface, they must accept a TCP connection in that interface and use its bi-directional nature to communicate data to that interface. Hence, each CR of a GO (say GO$_i$) must establish a TCP socket to every GO that is a client of GO$_i$.

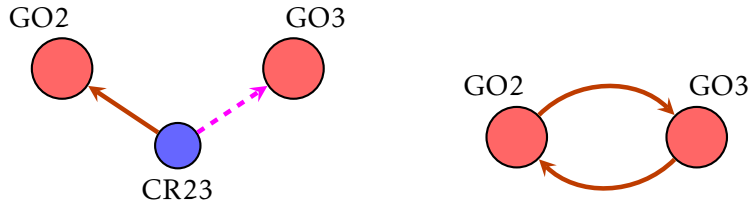
To adapt Figure 2.1 to reflect this variant, we can use TCP or UDP communication between pairs CL1–CR12, GO2–CR23, and GO3–CL3, and should use the following TCP connections: CR12 $\rightsquigarrow$ GO2 and CR23 $\rightsquigarrow$ GO3.

In Figure 2.12 we present a scenario with 3 groups in this variant.

Given that communication in GOCR$_{UC}$ is symmetric, for each direction it requires 1 IP unicast from the GO to its CR, and another from the CR to the next GO, resulting in 3 MAC unicasts over the GO channel. Therefore, $\text{SMAX} = WFS/3 = 18\ Mbps^*$, and $\text{SBDMAX} = WFS/6 = 9.0\ Mbps^*$.



a: GOCR$_{UC}$ scenario. b: GO2CR scenario.



c: GOCRGO scenario. d: GOGO scenario.

Figure 2.13: Experimental scenario for each tested topology.
Solid dark red and dashed pink arrows depict, respectively, WiFi and WFD interface connections in the arrow direction ($A \rightarrow B$, means A connects the interface to B).

GOCR$_{UC}$, then, uses WFD as *priority interface*, but keeps GOCR node configuration and data path, and only uses unicasts.

GOCR$_{UC}$ is then more efficient than GOCR, as it does not use broadcasts, which are slower and more energy demanding. This variant is, then, an improved version of GOCR and should present performance values better than the original version.

As a side note, if we revert to the case of having WiFi as *priority interface*, the GOCR$_{UC}$ variant or the GOCR topology can resort exclusively on unicasts, if each CR creates a TCP connection to its GO and the connection is used bi-directionally.

We can conclude that GOCR$_{UC}$ requires the use of TCP connections. To support the exclusive use of UDP unicasts it requires A5C devices.

Evaluation

Experiments report the communication speed computed from the round-trip of transmitting 100 MB between two GOs (GO2 and GO3) over TCP channels and using 1 KB data buffers. The exclusive use of TCP communication provides a homogeneous setting for all topologies, as TCP is required in GOCR$_{UC}$. All reported values are the average of 10 successful experiments conducted with a level of interference smaller than -80 dBm in the channels of the GOs. Test scenarios are similar to those of Figures 2.12, 2.2, 2.3 and 2.4, having UDP communications replaced by TCP connections. Test scenarios are then presented in Figure 2.13 and use the TCP connections presented in Table 2.6.

Table 2.6: Experimental setup.

Topology	TCP connections
$GOCR_{UC}$	GO2 ⇝ CR23, CR23 ⇝ GO3
GO2CR	GO2 ⇝ CR23, CR23 ⇝ GO3, GO3 ⇝ CR32, CR32 ⇝ GO2
GOCRGO	GO2 ⇝ CR23, CR23 ⇝ GO3
GOGO	GO2 ⇝ GO3

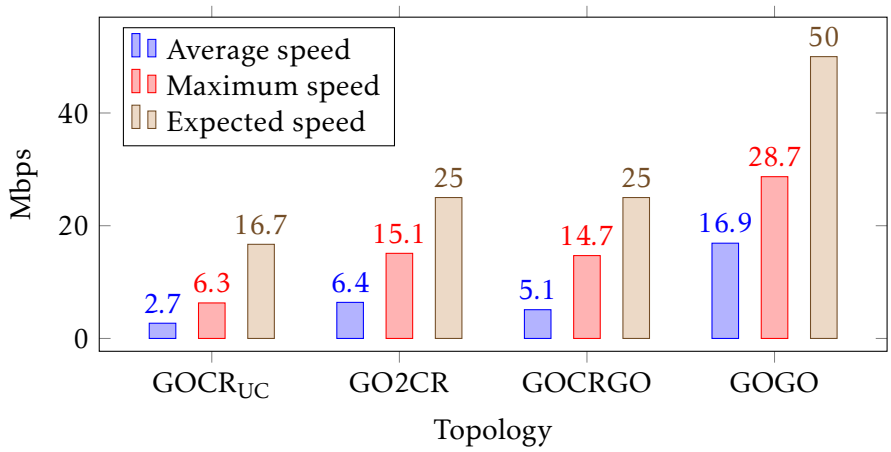


Figure 2.14: Topologies communication speed.

Topologies communication speed

Here we want to test the communication speed. Figure 2.14 depicts the resulting average, maximum and expected communication speed. The *maximum* relates to the maximum speed observed in 1 second period at reception in the final (and initial) node (GO2). The *average* conveys the average speed from the instant when the initial/final node (GO2) starts the test until it receives the last data byte. For the *expected* value, we considered the maximum speed $S_{MAX} = 100\ Mbps$, as the actual measured value was $103.8 Mbps$ in a TCP connection from a CL to its GO.

As depicted in Figure 2.14, the *maximum* and *average* communication speeds are considerably lower than the *expected* values. That is due to the fact that the communication speed slows down when devices use both interfaces simultaneously, making evident the hardware limitations of the smartphones and an inefficient use of absence control protocols: Notice of Absence Protocol (NoA) and Opportunistic Power Save Protocol (OPS) (see 2.1.1).

$GOCR_{UC}$ was clearly the worst performing topology of all. It should be noted that the performance of the GOCR would be far worse than $GOCR_{UC}$ as the expected speed of $GOCR_{UC}$ variant is $SBD_{MAX} = 16.7\ Mbps$ (WFS / 6), while the expected speed of GOCR topology is $SBD_{MAX} = 4.6\ Mbps$ (considering $WFS = 100\ Mbps$ and $BCS = 6\ Mbps$, as the measured broadcast speed (BCS) from a GO was 5.5 $Mbps$).

Between GO2CR and GOCRGO topologies, the former performed slightly better than

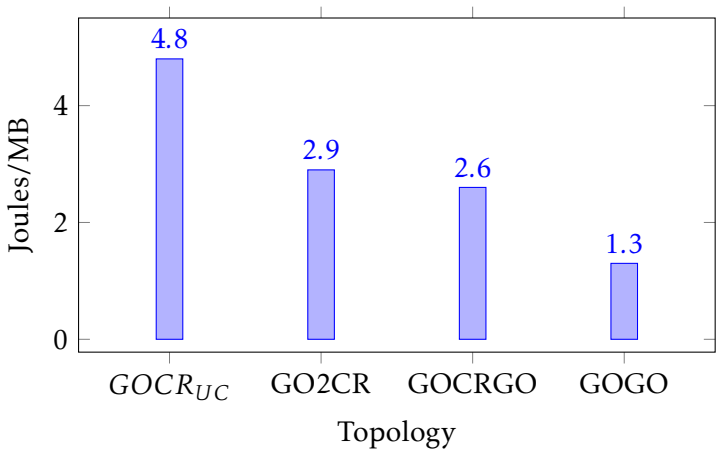


Figure 2.15: Topology consumed energy.

latter. That happens because in the former the load is distributed between the CRs, where each one of them carries the traffic in just one direction. In the latter, its unique CR has to carry the load in both directions.

The results confirm that GOGO is, as expected, the fastest topology, given that it requires fewer messages per GO channel.

Topologies consumed energy

Here we want to evaluate the consumed energy by each topology.

In tests we also extracted, every second, the instantaneous values for current and voltage from devices using the Android services, to get the consumed energy in the experiments.

Figure 2.15 contains the results for consumed energy in the experiments done for all tested topologies. Values are presented in $Joules/MB$.

Due to the differences between devices, to get more truthful values we subtracted their expected energy consumption at rest. We measured the energy at rest in $Joules/sec$ and subtracted a proportional value, to the time used in experiments, from the obtained values. All tests were performed with the screens turned on.

As expected, energy consumption depends mainly on the number of data transmissions.

Comparing GOCR topology and GOCR_{UC} variant: GOCR is expected to spend more energy than GOCR_{UC}, as the use of broadcasts requires more than 10 times the energy of unicasts. We measured the average energy required by a GO (Nexus 6) to send data, and the results are ($joules/MB$): 4.72 for broadcasts and 0.35 for UDP unicasts.

Another important measure is the *energy consumed per* WFR. Given that the GOCRGO and GO2CR scenarios cover 2 WFRS, we tested GOGO with 3 aligned GOs to cover that same distance, transferring only $1MB$. GOGO's average speed was 7.4 $Mbps$ and the energy consumed was 2.9 $joules/MB$. As the speed decreased significantly, the power

increased due to the difficulties in the communications, revealing the hardware limits of the devices when a GO is using both interfaces simultaneously. We thus conclude that GOGO (with 2.9 per 2 WFRS) is less energy efficient, per WFR, than GOCRGO (with 2.6 per 2 WFRS).

These values show that $GOCR_{UC}$ is the one that consumes more energy. Therefore, the GOCR topology should consume even more. Between the GOGO and GOCRGO topologies, the former consumed less energy per group, while the latter may consume less energy per distance (depending of the position of the CR nodes).

2.2.4 Conclusion

From the conducted experiments we observed that: a) GOGO is better than all other topologies when it comes to communication speed and energy per group; and b) GOCRGO is faster than $GOCR_{UC}$/GOCR and more energy efficient than the GO2CR when it comes to energy per distance.

The GOCRGO topology can be used to provide connection for LC–LC and LC–GO cases and the GOGO topology can do it for GO–GO, thus covering all situations.

The GOCRGO can be used to connect groups in mesh like structures and the GOGO can be used to complement those connections and create tree like structures. This allows us to use GOGO to extend groups, where client nodes also turn into GOs and have their own clients, creating extending group structures, called *clusters*. Clusters are then bigger structures than a group. They are managed by a central GO, called Dominant Group Owner (GOD). Clusters may then be interconnected by the use of GOCRGO, but also with GOGO when GODs can be used for interconnection.

We also proposed GO2CR, which needs two CR nodes between two GOs and can only use unicasts. We then proposed $GOCR_{UC}$, a variant for GOCR, which can be used with devices that have WFD as *priority interface* and can use TCP connections.

Finally, we conclude that proposed topologies present better expected and experimented values than the existing one (GOCR) and offer complementary ways to interconnect WFD groups.

2.3 Conclusions

In this chapter we first identified that the existing WFD intergroup connection and communication topology (GOCR) is not efficient for large-scale scenarios and then we proposed three topologies (GO2CR, GOCRGO and GOGO) to overcome existing limitations. These topologies can interconnect WFD groups in mesh and tree configurations and can only rely on unicasts.

We then analysed how these topologies should perform in relation to several aspects, concluding that developed topologies perform better than the existing topology. There

are some exceptions (GOGO needs A5C devices and GO2CR and GOCRGO have poor radio coverage per node), but these are minor issues.

We then proposed one variant ($GOCR_{UC}$) to enable the use of the existent topology (GOCR) by our mobile devices.

Finally, we evaluated the communication performance and energy consumption for all topologies and concluded that our developed topologies should perform better than the existing one.

Therefore, we affirmatively conclude that *"It is possible to do WFD intergroup connection and communication for large scale mesh networks"*.

Having WFD group interconnection solved, we will move to WFD network formation using these topologies.

Chapter 3

WiFi-Direct Network Formation

This chapter approaches the question *"Is it possible to systematically form WFD large scale mesh networks?"*.

In Section 3.1 we present current WFD network formation algorithms and identify their weaknesses in large scale scenarios. It also contains a review from network formation algorithms from nearby areas.

In Section 3.2 we present our newly proposed algorithm, called *RedMesh*, for WFD network formation in large scale scenarios. It also contains the algorithm evaluation, its limitations and reflections about possible improvements.

Finally, in Section 3.3 we expose our conclusions about the addressed research question.

3.1 Context and problems

Here we focus ons the context and problems in WFD network formation algorithms for large scale scenarios. In Section 3.1.1, we present the existing WFD network formation algorithms and showcase their problems. In Section 3.1.2, we survey network formation in the related areas of Bluetooth Scatternet Formation (BSF) and Mobile Ad Hoc Networks (MANETs) formation. Finally, in Section 3.1.3, we conclude the topic.

3.1.1 Existing WiFi-Direct network formation algorithms

To the best of our knowledge, only Baresi et al [5] and Casetti et al [9] presented algorithms to create networks with WFD off-the-shelf devices, with the simultaneous use of WiFi and WFD interfaces.

MAGNET network formation algorithm

The network formation algorithm in [5] (we name it MAGNET), is based on a communication topology (we name it MAGTOP), which uses unicasts and broadcasts and creates mesh group formations, by using CR nodes, i.e. LC/CL ones, among GOs.

In MAGTOP, CR nodes connect their WiFi and WFD interfaces to neighbour GOs, similarly to GOCRGO. However, unlike GOCRGO which can only use unicasts, the MAGTOP uses broadcasts to send data from the *non-priority interface*. As GOCRGO can also use broadcasts, we conclude that the MAGTOP is not a new topology, but a possible configuration of GOCRGO.

The MAGNET algorithm has the following stages:

1. *Neighbourhood discovering and GO election* where nodes advertise their existence and discover their neighbourhoods. Nodes advertise their intention value, calculated as a function of their processing power, battery life and intention to move. Nodes start in an *active* state. In a neighbourhood, nodes with the highest intention value, become GOs and move to the *capturing* state.

2. *Group creation* where GOs enslave up to 8, not slaved, neighbours. When a node finishes the capture it moves to an *inactive* state. Enslaved nodes also move to an *inactive* state. The *active* (unconnected) nodes with no *capturing* neighbours and with the highest intention value between *active* nodes become new GOs. Enslaved nodes inform their GO about neighbour groups they are aware of.

3. *Group interconnection* where each GO tries to connect to all neighbour groups by selecting the best local and remote nodes for interconnection. This is done in a global way running a Depth-First Search (DFS) algorithm to minimize node distance.

Hence, MAGNET algorithm uses a slower configuration of GOCRGO, as it uses one CR between GOs and uses broadcasts. The algorithm forms clusters with only GOs and their direct slaves and GOs try to connect to all their neighbour groups, possibly forming a mesh structure.

We conclude that this algorithm may be used to create large scale mesh networks. Yet, it has the following drawbacks:

1. it offers limited connection possibilities, as it uses only connections by CR nodes;

2. it uses clusters of depth 1, however, clusters of any depth create bigger clusters, which reduce the number of clusters and consequently the number of cluster interconnections needed; and

3. the greedy and local strategy to inter-connect all neighbour groups, may block connection to some of them and consequently create network segmentation. In our

evaluation, in Section 3.2.4.14 (Results from cluster connections with/without intermediation), we observed that connecting to all neighbour clusters is less efficient than just doing it to the ones that are not neighbours among them.

SGF network formation algorithm

Casetti [9] proposes the Smart-Group-Formation (SGF), a network formation algorithm that uses the GOCR communication topology and has the following stages:

1. *Neighbourhood discovering and GO election* where nodes acquire 2 hops away neighbourhood (neighbours of neighbours) information and then are elected as GOs the nodes that their (1-hop) neighbourhood cannot be covered by a larger neighbourhood (with more nodes) of another node;

2. *Group creation* where GO nodes enslave up to 8 not enslaved neighbours;

3. *Group interconnection* where each GO connects its WiFi interface to the GO with the highest rank value (depending on node resources and MAC address) in its neighbourhood and then accepts other GOs as slaves; and

4. *Relay node election* where each GO elects as relay node the slave with the highest rank value.

In this algorithm, each GO connects to another GO using its WiFi interface. If a GO has two neighbour GOs with higher rank values and these GOs are not themselves in radio range, it will connect to the highest, creating two separate tree segments that may never merge. Consequently, this algorithm may create several disjoint trees. Besides that, it does not take into account that GOs after the *Group creation* stage may not be able to accept all GOs that want to connect to them. Thus, the interconnection of groups may fail, creating unconnected groups.

Thus, we conclude that this algorithm is neither appropriate for creating large scale mesh nor tree networks.

3.1.2 Network formation algorithms from BSF and MANETS

Here we want to know if there are network formation algorithms, which can be used in large scale scenarios with WFD enabled devices, from the areas of Bluetooth Scatternet Formation (BSF) and Mobile Ad Hoc Networks (MANETs).

To enable the comparison of distinct technologies we establish a succinct way to highlight the differences between them. We denote as *masters:slaves* property, the number of masters and slaves that a node can have. The reference is the *masters:slaves* property of WFD nodes, which is *1:8* or *2:0*, whether they are GOs or not.

Network formation algorithms from BSF

Here we verify if the Bluetooth Scatternet Formation (BSF) may provide an algorithm to create large scale networks with WFD devices.

Bluetooth (BLT) is a wireless communication technology for short range, low energy consumption and reduced data rate exchanges. It organizes devices in groups, like WFD, that are named *piconets*. A set of interconnected *piconets* is called a *scatternet*. In a *piconet* the group head node is called master and the others nodes are called slaves, which corresponds to GO and CLs or LCs, respectively, in WFD. Similarly to WFD, a master node in BLT forwards data among group members.

BSF nodes have a *masters:slaves* property of *any:256*, as a node can have any number of masters. A master can have 256 slaves, but only 7 of them can be in the *active* state. Remaining slaves are in the *parked* state, meaning that they are registered in the master, but they cannot communicate in the group (piconet). They have to be *unparked*, to change to *active* state and to enable communication. Hence, masters have to manage their slaves, *parking* and *unparking* them.

To avoid parked nodes, which reduce the communication inside groups (piconets), some BSF algorithms only allow masters to have a maximum of 7 slaves. Therefore, in these algorithms we consider that BSF has a *masters:slaves* property of *any:7*. We found the following BSF algorithms that are *any:7* and form mesh networks: BlueMesh [41], BlueMIS I [59] and Eliminate [28]. However, as *any:7* is much more permissive than what WFD nodes can support, these algorithms cannot be used to form mesh networks with WFD devices.

There are also BSF algorithms that form tree networks. From them BSFWAVY-ODL [27] and BSFWAVY-ODL1 [27] are *any:7* and only BlueTrees-ODL [60] is *1:7*. So, BlueTrees-ODL is the only BSF algorithm that can be used to form WFD networks, but unfortunately it only forms tree networks.

We conclude that there is no BSF algorithm that respects *1:8* and *2:0* and can be used to form WFD mesh networks.

Network formation algorithms from MANETs

Here we check if the Mobile Ad Hoc Network (MANET), which is another related area to WFD, may provide an algorithm to create large scale networks with WFD devices.

In Mobile Ad Hoc Networks (MANETs) [24], nodes use the *ad hoc* radio mode and they are free to communicate with any node in their radio neighbourhood. Thus, in general, MANET nodes have a *masters:slaves* property of *any:any*. A master-slave relation, here, is just an organization relation, not a technological one. We only found the following MANET algorithms which limit the number of slaves to a number K, thus they have *any:K*, as they do not limit the number of masters: Flexible Weight Based Clustering Algorithm [16], Vote-Based Clustering Algorithm [32], Energy Conservation Cluster Algorithm [45], and Adaptive Multi-hop Clustering [38].

However, as *any:K* is not compatible with the WFD restrictions, we did not find any suitable MANET algorithm to be used with WFD devices.

3.1.3 Conclusion

In this section, we analysed the existing solutions to systematically form large scale networks of WFD devices. We saw that MAGNET is the only algorithm that can be used for that purpose. However, it has some drawbacks in order to achieve good connectivity in large scale scenarios, as it offers limited connection possibilities and uses clusters of depth 1 and a greedy strategy to connect neighbour clusters.

From the related areas of BSF and MANET formation we did not find any suitable algorithm to build WFD mesh networks.

So, we conclude that *"Is it possible to systematically form WFD mesh networks in large scale scenarios?"* is a research question that needs an effective solution.

3.2 Solutions and results, the RedMesh algorithm

Here we present our solution, the algorithm called *RedMesh* and its results for WFD network formation in large-scale scenarios with off-the-shelf devices.

In Section 3.2.1 we present some preliminary considerations for *RedMesh*. In Section 3.2.2 we introduce *RedMesh* and describe its stages. In Section 3.2.3 we present the algorithm auxiliary procedures. In Section 3.2.4 we evaluate the algorithm and compare it with the other existing algorithms (MAGNET and SGF). In Section 3.2.5 we present the connectivity limitations of the algorithm and present possible improvements. In Section 3.2.6 we reflect about possible improvements in the algorithm in relation to network redundancy and tolerance to node churn. Lastly, in Section 3.2.7 we present the final considerations about this research goal.

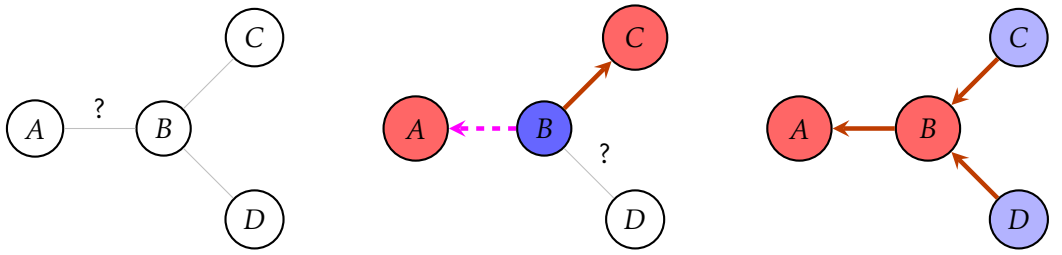
3.2.1 Algorithm preliminary considerations

Here we discuss and establish the base principles that will be used in the algorithm. We discuss: the use of groups or clusters; which interface should nodes use when enslaved; if we can just use the GOGO or the GOCRGO topology; and if we may use scenarios modelled by Unit-Disk Graphs (UDGs) and Arbitrary Graphs (AGs). At the end we present our conclusions.

3.2.1.1 Clusters or Groups

In a scenario we can use *groups*, which are formed by a GO and its slaves or we can use *clusters*, which are formed by one central GO and its slaves, and slaves of slaves of any depth. *Clusters* are then a group of *groups* connected together with any depth.

Groups are limited to 9 nodes, one GO and 8 slaves, while *clusters* do not have a limit, as they can have any depth. *Groups* will, then, be smaller structures than *clusters*, and



a: Radio visibility.

b: B connects by WFD interface.

c: B connects by WiFi interface.

Figure 3.1: Slave interface connection.
Red, dark blue and blue nodes depict, respectively, GOs, CRs and LC nodes. Solid brown and dashed pink arrows depict, respectively, WiFi and WFD interface connections in the arrow direction ($A \rightarrow B$, means A connects the interface to B).

consequently in scenarios with many nodes, they will have more *groups* than *clusters*. More *groups* will require more group interconnections.

We chose to use *clusters* firstly to minimize the number of interconnections and secondly because bigger structures will make network management easier (with less structures (*clusters*) to manage). One example of that simplicity occurs in routing: with less big structures (*clusters*) routing between then will probably require shorter route paths than when using small structures (*groups*).

Clusters usually use an initial GO, which we name Dominant Group Owner (GOD), that is elected as the node that ranks higher in its neighbourhood.

In our algorithm we use *clusters* and nodes that have a unique identifier (see Section 3.2.2, Algorithm introduction and stages) that is used to rank them.

3.2.1.2 Slave interface connection

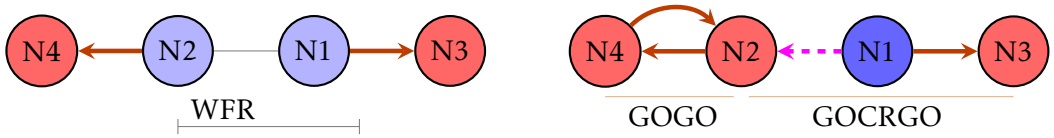
A node can be a slave (client) of a GO by either connecting its WiFi or WFD interface. If the node connects the WFD to the GO, the node may connect its WiFi interface to another GO. Whereas, if the node connects the WiFi interface to the GO, the node may connect the WFD interface to another GO, or it may create a group and provide connection for up to 8 nodes.

Figure 3.1 contains a scenario that we will use to discuss this choice. The figure contains nodes A, B, C and D, with the radio visibility shown in Figure 3.1a and node A is a GO that wants to enslave node B.

If node B connects the WFD interface to node A, node B can connect its WiFi interface to node C, but will not be able to provide connection for node D, creating two separate segments, as can be seen in Figure 3.1b.

If node B connects the WiFi interface to node A, node B can create a group and receive nodes A and D as clients, as can be seen in Figure 3.1c.

We conclude that nodes should be slaves by connecting their WiFi interface to the GOs, as in that way they have more future connection possibilities.



a: Initial scenario. b: Final scenario.

Figure 3.2: Connectivity scenario.
Red, dark blue and blue nodes depict, respectively, GOs, CRs and LC nodes. Solid brown and dashed pink arrows depict, respectively, WiFi and WFD interface connections in the arrow direction ($A \rightarrow B$, means A connects the interface to B).

3.2.1.3 GOGO or/and GOCRGO

Here we discuss the use of GOGO and/or GOCRGO topologies for group/cluster interconnection. Concretely, we want to know if they can offer connection for every situation. To make it simple, we only use clusters.

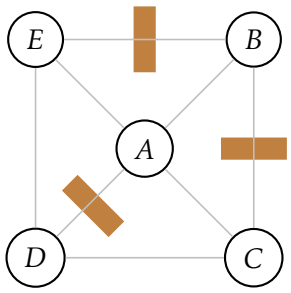
For both topologies we use the scenario in Figure 3.2a, which contains the nodes N4, N2, N1, and N3, positioned in line and in that order, where each node is in WiFi radio range of only its immediate left and right neighbour nodes (when they exist). Furthermore, the outer nodes N4 and N3 are dominant nodes, as they have the highest identifier in their neighbourhood and are then GOs. We also consider that N4 (GO4) enslaved N2 and that N3 (GO3) enslaved N1. As already seen, by default, a node is enslaved by another node, by connecting its WiFi interface to the other.

If we consider the exclusive use of the GOGO topology to connect these two clusters, N2 and N1 nodes should turn into GO nodes, as GOGO requires a path with only GO nodes, but as none of them have any free interface to connect to the other, they cannot be connected. Therefore, GOGO topology cannot offer connectivity in this case.

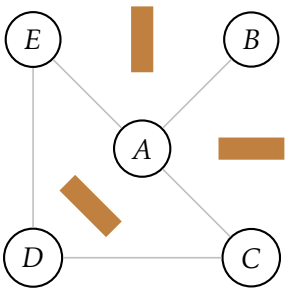
If we consider the exclusive use of the GOCRGO topology to connect these two clusters, we will need a path with a CR between two GOs. However, we cannot turn N1 into a GO, because in that case, N1 and N3 will require the GOGO topology to achieve connection between them. That equally happens if we try to change N2 into a GO. Thus, we conclude that the GOCRGO topology cannot offer connectivity in this case either.

However, the above scenario can become fully connected, having all nodes connected in a single network, if we use both topologies. We can achieve full connectivity if we start from the initial scenario, then N2 becomes a GO and finally N1 connects its WFD interface to N2. This way, GOGO topology is used between N4 and N2 and GOCRGO is used between N2, N1 and N3. To make the scenario complete, we need to have the GOGO topology well formed. That means that N4 has to connect its WiFi interface to N2. The resulting network can be seen in Figure 3.2b.

Therefore, GOGO and GOCRGO topologies provide connectivity in all situations (CL-CL and CL-GO with GOCRGO and GO-GO with GOGO) and their combined usage improves the chances of achieving full connectivity.



a: Scenario modelled as a UDG. Only range is considered.



b: Scenario modelled as an AG. Range and obstacles are considered.

Figure 3.3: Scenario modelled in a UDG and in an AG.
Gray lines depict the radio visibility. Brown rectangles depict obstacles.

WiFi-Direct scenarios modelled in Unit-Disk Graphs (UDGs) or/and in Arbitrary Graphs (AGs)

In general, scenarios can be based on Unit-Disk Graphs (UDGs) or Arbitrary Graphs (AGs). Here, we would like to know if WFD can ensure node/device connectivity in both types of scenarios. As we are mentioning graphs, we will use the term *nodes* instead of *devices*, but from now on they have the same meaning in this thesis. We will also use the terms master and slave, respectively, to denote a GO node and a client of a GO node.

In a scenario modelled as a Unit-Disk Graph (UDG) it is assumed that if two nodes are in range, they **will** be radio visible to each other.

However, the radio visibility is not only affected by range, as it can be reduced or cancelled by obstacles, radio interferences due to other nodes/devices, reflections (from its own node/device radio waves, or from waves from other nodes/devices), or by natural electromagnetic atmospheric phenomena. Thus, as scenarios modelled as UDGs do not consider all aspects that can affect radio visibility, they cannot be considered as very realistic.

In a scenario modelled as an Arbitrary Graph (AG) it is assumed that if two nodes are in range, they **may** be radio visible to each other. Therefore, nodes may be in radio range, but not visible by radio (unconnected in the radio visibility graph). That is what happens many times in reality, due to obstacles or other sources of radio interferences.

Thus, in a UDG two nodes in radio range are always visible, however, in an AG they may be visible or not. Hence, in the same scenario, any UDG model could be described by an AG model, but the opposite is not always true.

With $5 \leq L \leq 8$[1] we affirm that we can always connect all nodes in a UDG modelled scenario of WFD nodes in a single network. That occurs because in UDGs, 5 is the maximum number of independent sets of neighbour nodes of a node (u), that can cover all neighbour nodes of that node (u) [41, 59]. That kind of set is called a Maximal Independent Set (MIS). A MIS, or a set of nodes V_{mis}, is defined by: "Given a graph $G = (V, E)$, $V_{mis} \in V$ is

[1] L is the maximum number of slaves/clients that a master can have.

Algorithm 1 Maximal Independent Set (MIS) Calculation

```
1: function computeMIS(u)                                   ▷ u : node
2:     MIS ← ∅
3:     X ← N_f(u)\N(S(u))
4:     while X ≠ ∅ do
5:         n ← getNode(X)
6:         MIS ← MIS ∪ {n}
7:         X ← X \ ({n} ∪ N(n))
8:     end while
9:     return MIS
10: end function
```

independent if none of its nodes are neighbours to each other and it is maximal if V_{mis} is not a subset of any independent set of G.", from [27]. Here, V is the set of nodes in the scenario and E is the set of radio visibility links among those nodes. With $5 \leq L \leq 8$, we enable the formation of a MIS for all nodes (which requires $5 \leq L$) and respect the WFD *masters:slaves* restriction of *1:8* (which requires $L \leq 8$).

To connect all the nodes in a scenario we can use the following approach: consider one node as enslaved; and enslaved nodes enslave their not enslaved neighbours. This approach builds a tree with the all nodes. Nodes, however, should not enslave more than L nodes. Hence, we can use the following algorithm: promote node 0 to a master; masters enslave the MIS of their free (neither slave nor master) neighbour nodes; and enslaved nodes turn into masters. Nodes get the MIS of their free neighbour nodes set by executing the following actions until that set is empty: 1) select to MIS one node from free neighbour nodes set and 2) remove from free neighbour nodes set the selected node and all its neighbours. Algorithm 1 describes the function to get that MIS. In that algorithm: $N_f(u)$ is the set of free neighbour nodes of u, $S(u)$ is the set of slaves of u, getNode(X) is a function that returns one node from the set of nodes X and $N(u)$ is the set of neighbour nodes of node u or if u is a set, then $N(u)$ is the set of all neighbour nodes of all nodes in u. However, due to concurrency, a node from the MIS of a master may be enslaved by another master. When that happens, the former master should rebuild its MIS, using the updated version of N_f (without the nodes that it knows that are no longer free nodes, including its slaves) and without the neighbours of its slaves (grey action in the algorithm). This way, a node u will be enslaved for sure, if not enslaved before, by the last neighbour node to be a master. That happens, because that last neighbour master m has no other free neighbour node of u (and neighbour of m) that can be used to enslave u; therefore m must enslave u. Therefore, all nodes will be enslaved by some neighbour master and because maximal independent groups of neighbour nodes are not more than 5, we conclude that this procedure ensures that all nodes in the scenario will be connected in a tree with nodes that do not have more than 5 slaves/clients.

We thus conclude that using $5 \leq L \leq 8$ ensures connectivity in UDG modelled scenarios of WFD devices.

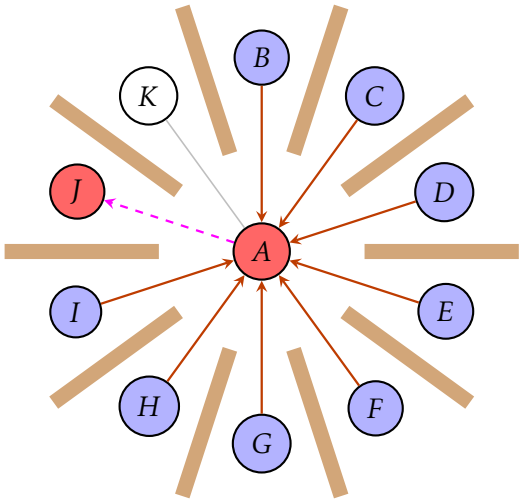


Figure 3.4: An AG modelled scenario not connected by WFD.
Solid dark red and dashed pink arrows depict, respectively, WiFi and WFD interface connections in the arrow direction ($A \rightarrow B$, means A connects the interface to B). Brown rectangles depict obstacles. Grey line depicts the radio visibility.

In relation to scenarios modelled as AGs, we affirm that a communication technology that has nodes with limited number of slaves and masters, cannot guarantee connection.

To prove that, we took into consideration the scenario in Figure 3.4 with: one node in the centre; 10 nodes around the central node and equally spaced; walls between the outer nodes that only enable communication to the central node; and nodes are WFD nodes and only accept 8 slaves (in the WFD interface) and can only have one master (connected by the WiFi interface). In this scenario, the central node can only connect to 9 nodes, 8 as slaves by the WFD interface and one as master by be WiFi interface. Therefore, in the figure, node K cannot be connected and we conclude that it is not possible to achieve connection in the scenario.

In general, a communication technology that has nodes with limited number of slaves and masters, nodes will not achieve connection in AG scenarios with cases of nodes with more neighbours, not visible among themselves, than the sum of the number of slaves and masters that a node can have.

We then conclude that to ensure connection, we should use WFD in scenarios modelled by a Unit-Disk Graph (UDG).

3.2.1.4 Conclusions of preliminary considerations

We will use cluster instead or groups, in order to reduce cluster interconnections. Nodes will be enslaved by connecting its WiFi interface to the GO.

We will use both GOGO and GOCRGO topologies. We will use the GOGO topology inside clusters to provide cluster extension. We use this topology because it has a better radio coverage per node than GOCRGO topology. We will use the GOCRGO and the GOGO topology for cluster interconnection. However, we will use the GOGO topology

only when the dominant node can be used for cluster interconnection. In those cases, the dominant node will connect its WiFi interface to a node in the other cluster.

3.2.2 Algorithm introduction and stages

The *RedMesh* algorithm should systematically form mesh networks that interconnect WFD-enabled off-the-shelf mobile devices in small to large-scale scenarios, establishing connections that may only use unicast (TCP or UDP) communication. The main design goal of the algorithm is to maximize network connectivity.

The algorithm has the following requisites: nodes have an immutable and unique identifier, nodes are static, nodes are A5C, all nodes start the algorithm at the same time, nodes do not fail, communication does not have any interferences, and scenarios are modelled as Unit-Disk Graphs (UDGs).

Node identifiers should be unique and manifest the intention of a node to be a GOD, a GO or a LC node, with higher values for a GOD and successively lower values for a GO and with the lowest values for a LC node. The identifier can be a metric that considers node resources, like battery power available, CPU, memory available, intention to move and complemented with the MAC address to break ties. The metric based on the node resource availability enables to rank them by current characteristics, avoiding, for example, the selection of a node almost empty of battery to be a GOD. However, in this thesis we do not consider changes in the identifiers over time and therefore they are immutable.

The algorithm should start with WFD groups, then extend those groups to form clusters and finally do cluster interconnection. The networks formed should provide TCP connections or enable UDP datagrams between neighbour connected nodes and will require an addressing and a routing scheme on top of them.

The algorithm applies the following sequence of stages, which will be described in the next subsections:

1. Dominant Node Election (DNE);
2. Cluster Building (CLB);
3. Cluster Neighbourhood Gathering (CNG);
4. Main Cluster Interconnection (MCI); and
5. Final Cluster Interconnection (FCI).

Before describing each stage, we introduce some notation and base definitions. Nodes are ranged over by u, v and w and have a unique identifier denoted by $id(u) \in \mathit{NodeIdentifier}$, being *NodeIdentifier* a totally ordered set. A set of nodes is denoted by U. The set of neighbours of u is denoted by $N(u)$, being the notation extended to sets: $N(U) =$

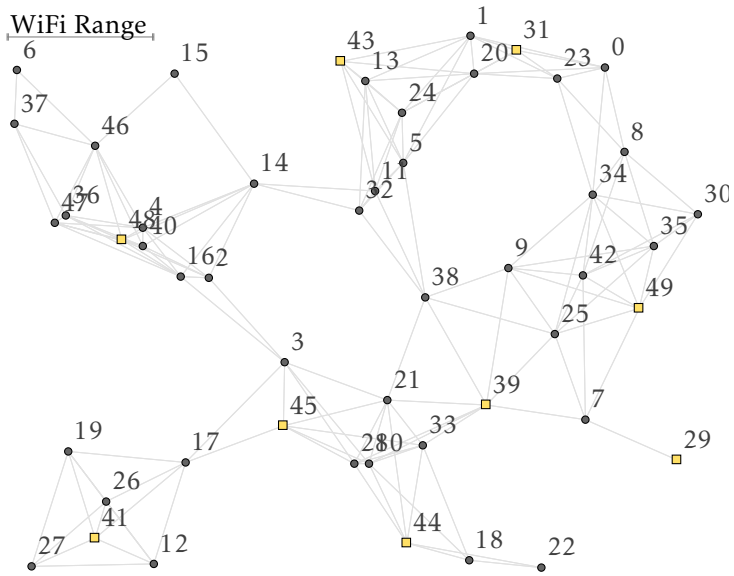


Figure 3.5: Running scenario after DNE stage.
Yellow squares, grey circles and grey lines depict, respectively, dominant nodes (GODs), other nodes, and radio visibility.

$\bigcup_{u \in U} N(u)$. The set of neighbours of u with identifier lower than u is given by $N_s(u) = \{v \in N(u) \mid id(v) < id(u)\}$. The set of slaves of u is denoted by $S(u)$.

Lastly, we use L as the usual maximum number of slaves allowed per GOs. So, as $L \in [5, 8] \subset \mathbb{N}$, we will set $L = 5$ as it is the most restrictive case. Neighbourhood coverage is obtained by computing the MIS [59] of the radio visibility network of the neighbourhood.

3.2.2.1 Dominant Node Election (DNE) stage

In the first stage of the algorithm, Dominant Node Election (DNE), every node u begins by discovering its neighbours ($N(u)$) and the neighbours of those neighbours, which is the 2 hops away neighbourhood ($N(N(u))$).

For that purpose, u initializes its WFD interface and announces itself broadcasting its identifier via BAS[2]. Once nodes discover all their direct neighbours, they broadcast that information, allowing nodes to compute their second level (2 hops away) neighbourhood. From their direct neighbours, each node can individually determine if it is a *dominant* node, i.e., if it is the node with the highest *id* in its neighbourhood. Dominant nodes then turn into GOs and we call them Dominant Group Owners (GODs).

Figure 3.5 depicts the result of this process for a 50 node scenario that we will use as a running example. In the figure we can see 9 GODs (yellow squares), which are the only nodes that have the highest *id* in their neighbourhood.

[2] All broadcasts are BAS broadcasts and carry the *RedMesh* service tag to enable system identification.

Algorithm 2 Candidate Slave Set Calculation

```
1:  function computeCSS(u, EBO)    ▷ u : node, EBO : set of nodes enslaved by others
2:      CSS ← Ø
3:      Candidates ← N_s(u) \ (EBO ∪ S(u) ∪ N(S(u)))
4:      while Candidates ≠ Ø do
5:          n ← getHighestRankedNode(Candidates)
6:          CSS ← CSS ∪ {n}
7:          Candidates ← Candidates \ ({n} ∪ N(n))
8:      end while
9:      Candidates ← N_s(u) \ (EBO ∪ S(u) ∪ CSS)
10:     while #CSS < L and Candidates ≠ Ø do
11:         n ← getHighestRankedNode(Candidates)
12:         CSS ← CSS ∪ {n}
13:         Candidates ← Candidates \ {n}
14:     end while
15:     return CSS
16: end function
```

3.2.2.2 Cluster Building (CLB) stage

The Cluster Building (CLB) stage starts on GODs and builds their clusters.

GODs trigger this stage by trying to enslave their neighbours. Once enslaved, nodes turn into GOs and then try to enslave their lower (*id*) neighbours, causing the enslaving process to flow downwards from higher to lower *id* nodes.

Each GO will: 1) ensure that all lower neighbours are enslaved by a node; and 2) try to enslave the maximum possible number of lower neighbours, which is L. With 1) we ensure connectivity, that is, all lower neighbours will be connected by a node. Therefore, after this stage, all nodes are grouped into clusters dominated by GODs. With 2), we maximize the number of slaves, hence reducing the number of GOs. Fewer GOs, as fewer masters in BLT require less radio transmissions, enable faster routing (as there are less groups to cross), reduce radio interferences and network maintenance costs [27, 32]. Consequently, better and more energy efficient networks will be formed.

To achieve the goals of both 1) and 2), we start with a MIS and complement it with more nodes to fulfil the L quota. So, for a GOD u, we get from the set of lower neighbour nodes ($N_s(u)$) the set of L highest neighbour nodes that can enslave all remaining nodes in $N_s(u)$. We call this set Candidate Slave Set (CSS).

However, due to concurrency, the enslaving process may fail. Thus, each time a GOD receives an enslave rejection, or perceives from BAS traffic that a CSS node has already been enslaved, it stores those nodes in a set called Enslaved By Others Set (EBO) and rebuilds the CSS. CSS successfully enslaved nodes are removed from CSS.

Algorithm 2 presents the function to compute CSS. It begins by setting CSS to a MIS of $N_s(u)$ (lines 2 to 8), excluding Enslaved By Others Set (EBO), $S(u)$ and $N(S(u))$ nodes ($S(u)$ is initially empty). This part is similar to the base MIS actions presented in

Algorithm 1 (Section 3.2.1.3), but here we want to produce a descendant wave, so nodes only take in consideration their lower neighbours (those with lower *ids*) and from them they select the higher nodes first. Thus, GETHIGHESTRANKEDNODE(X) is a function that returns the node with the highest rank from the set of nodes X. Next, the algorithm increases CSS with the remaining highest ranked $N_s(u)$ nodes (lines 9 to 14) to fulfil the L quota. When the enslaving process ends ($CSS = \emptyset$), the nodes without slaves revert from being a LC/GO to being just a LC. At the end, every node broadcasts its state as a tuple

$$\langle u, GOD, isGO, WFD_{GO}, WF_{GO}, S(u), N(u) \rangle$$

where u is the node *id*, GOD is the *id* of the dominant node of u, $isGO$ is true if u is GO, WFD_{GO} and WF_{GO} are the *ids* of the connected GOs by those respective interfaces of u; and $S(u)$ and $N(u)$ are respectively the slave and neighbour list of u. Nodes also broadcast that tuple every time they change their state.

When this stage ends, each GO u has a maximum of L slaves and every node $v \in N_s(u)$ is a slave of u or in range of another GO w, with $id(v) < id(w) < id(u)$. The reflexive transitive closure of the slavery relation S applied to a GOD u, i.e. the closure of $S(u)$, forms what we call a *cluster*. From the perspective of a cluster, nodes belonging to the cluster are called *local nodes*, while the others are called *remote nodes*. All slave to master (GOs) connections use the WiFi interface, leaving the WFD interface free to allow these nodes to either become GOs themselves or to connect to some other GO, maximizing future connection opportunities. Figure 3.6 illustrates our running example at the end of this stage. In the figure we can see the 9 GODs, their clusters and the WiFi connections of the GODs (when they have a GO slave).

3.2.2.3 Cluster Neighbourhood Gathering (CNG) stage

The goal of the Cluster Neighbourhood Gathering (CNG) stage is fourfold:

1. send node-level neighbourhood information to GOs;
2. identify the nodes that can interconnect clusters;
3. promote some of such nodes to GOs; and
4. send cluster-level neighbourhood information to higher neighbour clusters.

Firstly, we clarify that nodes whose neighbourhood extends beyond the boundaries of their own cluster and reaches nodes from other clusters, may be used for cluster interconnection and are called candidate *gateway* nodes. Besides that, from the perceptive of a cluster, a *local* candidate *gateway* node is a candidate *gateway* that belongs to the cluster and a *remote* candidate *gateway* node is a candidate *gateway* from another cluster.

Nodes start this stage upon the reception of the final broadcasts from the previous stage (CLB). Then, LC nodes send local and received CLB information to their GOs.

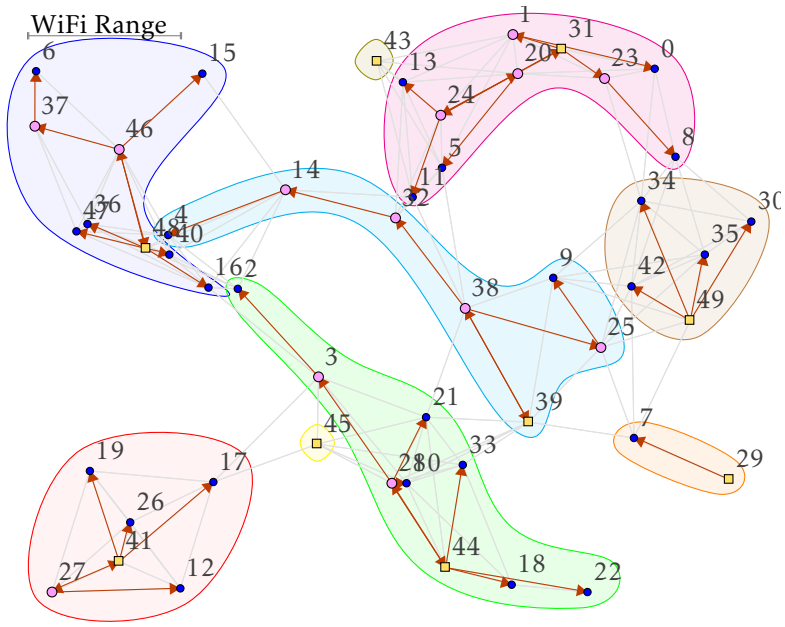


Figure 3.6: Running scenario after CLB stage.
Circular lines delimit clusters. Yellow squares, larger pink/lighter circles and smaller blue/darker circles depict, respectively, dominant nodes (GODs), LC/GO nodes and LC nodes. Brown arrows $u \rightarrow v$ are master-slave relations from u to v nodes, where v connects its WiFi interface to u. Grey lines are the radio visibility.

GOs then combine their own local information with the one received from all slaves and propagate it to their own GOs. GODs will finish this step knowing the node state, neighbourhood information from all local nodes and from remote candidate gateways in range and, also, their neighbour clusters (the *ids* of their GODs). With this information, they build a table, called *visibility table*, with all pairs of possible local and remote candidate gateways.

GODs then proceed with the selection of local candidate gateways that are the only nodes able to interconnect two or more neighbour clusters. These nodes are then promoted to GOs, becoming LC/GO nodes and called LCsToGOs. This step is of the utmost importance as it prevents higher clusters – with higher ranking GODs – to ask these gateway LCs to connect to them, transforming LC nodes into CR ones and precluding them from connecting to any other cluster. It is then, a preparatory measure to prepare connection points for neighbour clusters.

Lastly, gateway and LCsToGOs information is propagated upwards from lower to higher GODs. These messages are sent, between GODs, from the source GOD to the local *gateway* (on the way to the destination GOD and obtained from the *visibility table*), then sent to remote *gateway* and finally addressed to the remote GOD. So, GODs will know the complete neighbourhood information, which includes LCsToGOs, from lower neighbour clusters. That information enables them to start cluster interconnection in a descending round.

In all the algorithm, communication between unconnected nodes goes by broadcasts (BAS) and otherwise by unicasts.

3.2.2.4 Main Cluster Interconnection (MCI) stage

The Main Cluster Interconnection (MCI) stage is where more cluster interconnections occur. It starts in the highest GODs, among neighbour clusters, and goes down until the lowest clusters. Thus, it is a descending round. GODs start it by receiving MCI-end notifications from all their higher neighbour clusters and updating the *visibility table* with that information.

Then, they put in *cltsToConnect* list the *ids* of all their lower neighbour GODs/clusters and in a descending order. Next, they use Cluster Interconnection Procedure (CIP), described in Section 3.2.3.1, to interconnect those clusters following that order. After each cluster interconnection, GODs update the *visibility table* accordingly. However, before each interconnection, GODs check if there are LCs that are the unique nodes to connect more than one remote cluster and, if so, they turn them into LCsToGOs (LC/GO). This is necessary to avoid local gateway LC nodes to act as CR nodes and to use the WFD interface to connect to only one cluster, when they must connect to two or more. This is, thus, a second round of LCsToGOs.

After processing all *cltsToConnect*, GODs check which unconnected clusters can be connected by an intermediate path of clusters and lastly send MCI-end notifications to all neighbour lower clusters. This is done in End of Interconnections Procedure (EIP), which is described in Section 3.2.3.2. A MCI-end notification contains cluster connections done and achievable through intermediation, and cluster interconnections that cannot be done.

Figure 3.7 shows our running scenario after this stage. It highlights the cluster interconnections done in thick arrows. There are 4 interconnections by WFD: $14 \rightarrow 40$ (by R1 rule, see Section 3.2.3.1), $3 \rightarrow 16$ (R1), $7 \rightarrow 42$ (R2) and $23 \rightarrow 34$ (R1), and 3 by WiFi: $3 \rightarrow 45$ (R3), $20 \rightarrow 43$ (R3) and $25 \rightarrow 49$ (R3).

3.2.2.5 Final Cluster Interconnection (FCI) stage

The Final Cluster Interconnection (FCI) stage starts when the lowest GODs, between neighbour ones, receive MCI-end notifications from all their higher neighbour clusters. It starts on other GODs when they receive FCI-end notifications from all their lower neighbour clusters. This stage will then flow upwards from the lowest to the highest clusters and it is, thus, an ascending round.

GODs put higher clusters to connect, sorted ascendantly, in *cltsToConnect* list and execute again Cluster Interconnection Procedure (CIP) over the list.

After that, GODs run again the End of Interconnections Procedure (EIP), using all the data from the MCI and FCI stages, making a final decision about which clusters are connected directly by an intermediate sequence of clusters or not connected at all. Once concluded, they send their FCI-end notifications to their higher neighbour clusters.

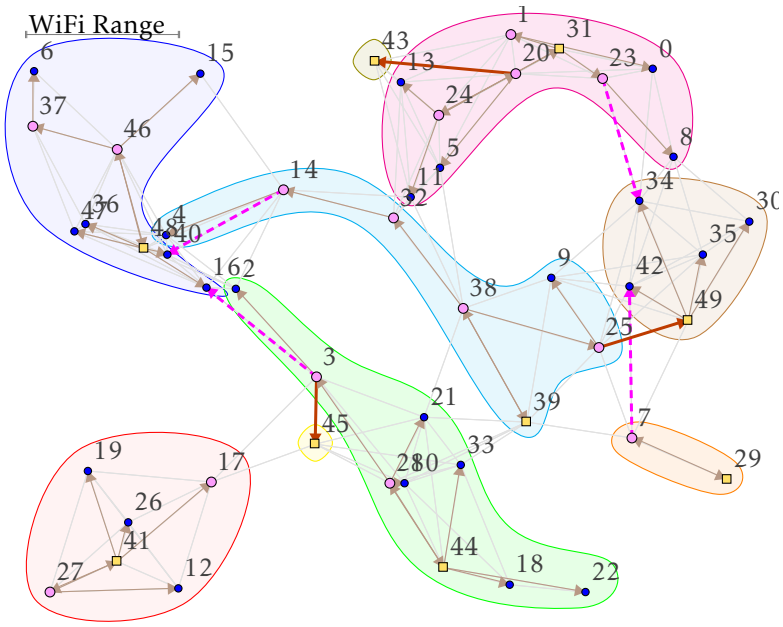


Figure 3.7: Running scenario after MCI stage.
Circular lines delimit clusters. Yellow squares, larger pink/lighter circles and smaller blue/darker circles depict, respectively, dominant nodes (GODs), LC/GO nodes and LC nodes. Brown (thin and thick) arrows $u \rightarrow v$ are master-slave relations from u to v nodes, where v connects its WiFi interface to u. Pink dashed arrows are WFD connections, similar to WiFi ones. Thick arrows, brown and pink, are cluster-interconnections. Grey lines are the radio visibility.

In our running scenario, this stage only connected nodes 3 and 17 ($3 \rightarrow 17$), as shown in Figure 3.8. Node 17 stopped being a GO and connected its WFD interface to node 3, using R5 interconnection rule (see Section 3.2.3.1, Cluster Interconnection Procedure (CIP)). Therefore, the application of this final stage created just one cluster interconnection. However, it enabled to conclude the formation process in just a single connected network.

3.2.3 Algorithm auxiliary procedures

The algorithm has the following auxiliary procedures or managements that will then be described: Cluster Interconnection Procedure (CIP), End of Interconnections Procedure (EIP), WiFi Management, and Routing Management.

3.2.3.1 Cluster Interconnection Procedure (CIP)

Due to the highly restrictive nature of WFD, we designed the *RedMesh* algorithm aiming at maximizing the overall connectivity. To achieve this goal we minimized the cluster interconnections done to the minimum necessary. This way, we maximize the possibilities of success for the really necessary cluster interconnections.

In this procedure, GODs in MCI and FCI stages try to connect to all the clusters in the *cltsToConnect* list. They do it sequentially, by the order the clusters are in the list and

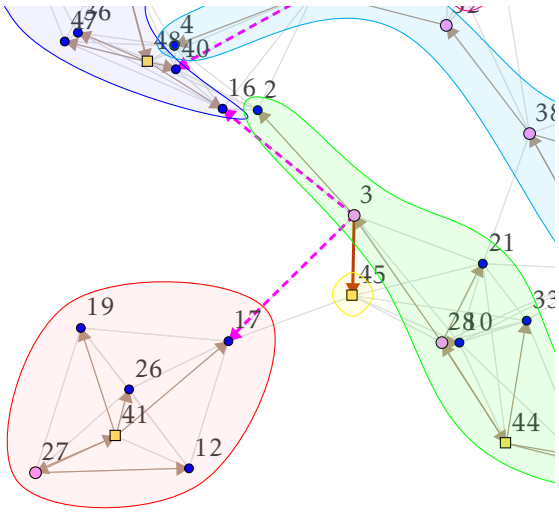


Figure 3.8: Running scenario after FCI stage.
Focused on the new cluster interconnection: 3 → 17 (R5). Circular lines delimit clusters. Yellow squares, larger pink/lighter circles and smaller blue/darker circles depict, respectively, dominant nodes (GODs), LC/GO nodes and LC nodes. Brown (thin and thick) arrows $u \rightarrow v$ are master-slave relations from u to v nodes, where v connects its WiFi interface to u. Pink dashed arrows are WFD connections, similar to WiFi ones. Thick arrows, brown and pink, are cluster-interconnections. Grey lines are the radio visibility.

until the list is empty. For each cluster, firstly GODs try to get a connected intermediate path through neighbour clusters, using the information received from MCI-end and FCI-end (if available) notifications and local connections already established. If such path is found, the cluster is moved to the *intermedClts* set and the path is stored in a map called *intermedPaths* in pairs with ⟨destination, intermediation⟩. If there is no known connected path to that cluster, GODs proceed to build such path. Then they try to get a cluster interconnection rule that matches an existing pair of local and remote (from the desired cluster) candidate gateways. To achieve that, they use the *visibility table* and the rules presented in Table 3.1 and described next in *Cluster Interconnection Rules*. If no rule is found, the cluster is placed in the *notConnClts* set, otherwise GODs execute the actions of the interconnection rule and, in case of success, the cluster is moved to *connClts* set. If the interconnection rule fails, GODs update the *visibility table*, with received information and repeat this procedure for the same cluster.

Now, we will describe the *Cluster Interconnection Rules*, the *Cluster Interconnection Rules Pair Selection*, the *Cluster Interconnection Generic Rule Actions* and the *Cluster Interconnection Delegation Rule (R5) Actions*.

Cluster interconnection rules

We use five interconnection rules, named R1, . . . , R5, summarized in Table 3.1. The table presents for each rule: the name, the initial and final states for the local and remote gateways and the connection details (WFD or WiFi and direction). Rules R1, R3 and R4 are applied firstly in the version/direction presented in the table, but they are also

Table 3.1: RedMesh cluster interconnection rules.

	Initial configuration		**Final configuration**		
Rule	local	remote	local	conn	remote
R1	LC	LC/GO	LC/CL	⇠--	LC/GO
R2	LC	LC	LC/GO	--⇢	LC/CL
R3	GOD^{FL}	LC/GO	LC/GOD	⟵	LC/GO
R4	GOD^{FL}	LC	LC/GOD	⟵	LC/GO
R5	LC/GO^{D}	LC/GO	LC/CL	⇠--	LC/GO

Notes: ^{FL}, WiFi free or locally connected (in cluster); ^{D}, can delegate all slaves; A ⟶ B / A --⇢ B, master slave relation from A to B and B connects WiFi / WFD interface, respectively, to A.

applied in the reverse direction if the presented version failed. For each Ri for i ∈ {1, 3, 4}, we denote the version presented in the table by Ria and the reverse by Rib, e.g., for R1, the initial configuration of R1a is local = LC and remote = LC/GO, whilst in R1b it is local = LC/GO and remote = LC. The algorithm first applies the *a* rule versions, as we usually favour remote GOs over local ones, because a remote GO may be used by the remote cluster to connect to a third cluster or receive other nodes as slaves (R5). The procedure returns the first rule version that finds a matching pair of nodes. The default rule testing order is R3 :: R1 :: R2 :: R4 :: R5, which was obtained empirically from extensive evaluation.

Concerning rule details, in **R1**, LC gateways connect their WFD interface to a LC/GO gateway in the other cluster. In **R2**, local LC gateways turn into LC/GO and receive a WFD connection from a remote LC. **R3** connects GODs WiFi interface to a LC/GO gateway (that supports one more slave) in the other cluster. The WiFi interface of the GODs must be free or locally connected, i.e. connected to a node in the same cluster. If connected, GODs must disconnect it first. **R4** connects the WiFi interface of the GODs to a LC gateway in the target cluster. R4 complements R3 and requires the LC gateway to firstly change into a LC/GO. R3b and R4b accept remote GODs with any WiFi interface configuration, as the local (in charge) GOD may be unaware of what happened meanwhile to those nodes. Remote GODs reject these rules (R3b and R4b) when they have the WiFi interface already connected to a gateway in another cluster. Consequently, local GODs, once having received those rejections, disable future selection of these rules for the same nodes. **R5** requires a local LC/GO gateway that can delegate all its slaves and a remote LC/GO gateway that can receive one more slave. The local gateway must delegate all slaves only to local nodes that are GOs or are LCs that can be turned into GOs. Then, it should stop being GO and connect its WFD interface to the remote gateway.

Cluster interconnection rules pair selection

A cluster interconnection rule may match more than one pair of gateways. Each rule applies, then, a criterion of pair selection to elect the most favourable one. The criterion consists of a series of clauses, electing the pair that consecutively verifies more of them

(based on appearance order). The selection clauses for each rule follows. **R1**: LC neighbour of fewer networks[3]; LC/GO with fewer slaves; higher (*id*) LC; higher LC/GO. **R2**: remote LC neighbour of fewer networks[3]; higher local LC; higher remote LC. **R3**: LC/GO with fewer number of slaves; higher LC/GO. **R4**: LC neighbour of fewer networks[3]; higher LC. **R5**: local GO that can delegate all slaves; if local GO has the local GOD as a WiFi slave, there must be a receptor for the WiFi interface of the GOD; local GO with fewer slaves; remote GO with fewer slaves; higher local GO; higher remote GO.

Cluster interconnection generic rule actions

A cluster interconnection rule starts with the GOD notifying the local gateway of the selected interconnection rule and pair of gateways. The local gateway then performs the necessary preliminary actions, like turning into a GO or disconnecting its WiFi interface, and subsequently connecting to the remote node or asking the latter to connect to it. Communication between gateways is done by broadcast (BAS). Finally, the local gateway sends the confirmation or rejection to its GOD. The GOD receives the final message and updates its *visibility table* with the success/rejection information. In case of success, the remote gateway also sends a message to its GOD, so that the routing information is updated.

Cluster interconnection delegation rule (R5) actions

Here, R5, the delegation rule, due to its complexity, is further detailed. We refer to the node that delegates its slaves as the *delegator*, a delegated node as a *delegatee* and a node that receive *delegatees* as a *receptor*. On rule selection, R5, besides the pair of selected gateways, also returns a set of delegations (*delegatee*/*receptor*) and the receptor of the WiFi interface of the GOD if it is connected to the *delegator*.

The first step of the rule is to change the WiFi interface of the GOD, if it is connected to the *delegator*. Subsequently, if there are *receptors* that are LC nodes, GOD notifies them to be GOs. After that, GOD sends the rule information (R5, gateways and delegations) to the *delegator*. The latter notifies all *delegatees* about their *receptors*, causing the *delegatees* to disconnect their WiFi interface from the *delegator* and to connect it to their *receptor*. With all *delegatees* disconnected, the *delegator* stops being a GO, connects its WFD interface to the remote node and then sends a message to its GOD with the success of the operation, deprecating all routing information concerning its old slaves. When *receptors* receive all connections from *delegatees*, they send a message to the GOD. Those messages also update routing information. Upon the reception of all messages from the *receptors* and the *delegator*, GOD updates its *visibility table* and finishes the application of the rule.

3.2.3.2 End of Interconnections Procedure (EIP)

This procedure executes the MCI and FCI stages final actions. First, GODs try to get an intermediate path to each cluster in *notConnClts* from all existing connections. If a path is

[3] Information from CNG stage and only considered in MCI stage.

found, the cluster is placed in *intermedClts* and the path in *intermedPaths*. Otherwise, the cluster remains in *notConnClts* set. Then, GODs send an end of stage message to all their lower/higher neighbour clusters, in MCI/FCI stages respectively, and conclude the stage. They use *intermedPaths* and the *visibility table* to get a path to those neighbour clusters that are in *intermedClts* and *notConnClts*, respectively. Messages carry connections done, intermediated and not done.

3.2.3.3 WiFi management

Two GOs may communicate directly by unicast, if both have the WiFi interface connected and one have it connected to the other. All GOs, that are not GODs, have their WiFi interface connected to another GO even before they are GOs. GODs are the only GOs that start with the WiFi interface disconnected. However, they connect it to their first GO slave. GODs may also have the WiFi interface disconnected when executing R3, R4 and R5 interconnection rules. In these cases, they notify their GO slaves to suspend unicast traffic and, when they reconnect their WiFi interface, they notify their slaves again to resume traffic. That traffic comes from intermediate cluster connections and routing purpose messages. A GO, non-GOD, may have the WiFi interface momentarily disconnected in the execution of R5 interconnection rule if it is a *delegatee* node. In these cases, traffic is also suspended between the disconnection, of the WiFi interface, from the *delegator* and the connection to the *receptor*.

3.2.3.4 Routing management

Routing is set inside clusters mainly in CLB and CNG stages and between clusters in MCI and FCI stages.

In CLB stage, nodes set route information from master-slave relations. In CNG stage, routing is obtained from messages that are sent to GODs. Nodes that receive those messages set the message source and other mentioned nodes as targets in WFD interface and the node that sent the message as their next hop. Besides that, GODs also register their WiFi connection, when connected. The routing information is registered in triplets: ⟨ destination, next node/hop, interface ⟩.

The MCI and FCI stages update routing when clusters are interconnected and intermediate paths to clusters are found. Cluster interconnections may have preparatory (e.g. changing WiFi interface or delegations) and final changes that require updates in routing information. On cluster interconnection completion, gateways in local and remote clusters set the other gateway as the next hop to reach the GOD of the other cluster, and update local routing by sending a message to their GOD. Consequently, GODs will be able to send unicasts to GODs of connected clusters, setting them as the final address and setting the *nextNode* and *interface* fields of the routing registry for the desired target.

For intermediate connected clusters, GODs send messages with an intermediation path, named *godsPath*, with the necessary GODs to reach destination. When a GOD

receives a message with *godsPath* and it is not the final GOD in the path, it removes itself from the path and relays the message to the next GOD. So, routing information is then composed of triplets: ⟨ final address, next hop, *godsPath* ⟩.

In general, a non-GOD node routes messages accordingly to the local next hop information. If the next hop is not available, the message is sent to the local GOD. When cluster changes occur, like in R3a, R4a and R5, nodes must queue in transit messages. When the cluster is stabilized, nodes will then route queued messages.

Therefore, messages flow via broadcasts (BAS) between unconnected nodes/clusters and via unicasts inside and between connected clusters.

3.2.4 Algorithm evaluation

This section evaluates *RedMesh* in scenarios with up to 250 nodes. It uses $L = 5$, because if we achieve good connectivity with such a small L value, we will, for sure, achieve it for larger L values as well. It contains a description of used simulator and scenarios, and the results obtained from each algorithm stage and from the comparison with existing WFD network formation algorithms.

3.2.4.1 Simulator used

Aiming to enable the reproducibility of our algorithm in medium- and large-scale scenarios, up to 250 nodes, and in a controlled environment, we opted for using the high-level network simulator WiDiSi [4][4]. This simulator is based on PeerSim simulator [37] and was also used in Baresi et al [5]. We modified the simulator to make it suitable to our needs. The most significant changes were to enable GO–GO and CR–GO connection and communication. We also configured WiDiSi to avoid message loss and changed the event management to keep message order. This way, we simulated TCP like conditions in node-to-node communication. Therefore, the results should be seen as the absolute best performance, as they do not take into account neither radio interferences, nor loss or out of order messages.

3.2.4.2 Scenarios used

The used scenarios are fully connected with relation to WiFi/WFD radio visibility. All nodes are, then, reachable by radio either directly or indirectly by way of other reachable nodes. Thus, in one scenario, starting from any node we can reach any other node in the scenario.

The scenario generation algorithm used employs two sets, the *end* set that will contain the nodes placed in the scenario and the *free* set that will contain the nodes that must be placed in the scenario. The algorithm has the following steps:

1. place node 0 (with *id* 0), positioned in scenario centre, in *end* set;

[4] Code location: https://github.com/deib-polimi/WiDiSi, accessed on Jul 26, 2019.

Table 3.2: Scenarios metrics

	Scenarios size (# of nodes)				
Metric	**50**	**100**	**150**	**200**	**250**
With1Neig	2.68	3.16	3.50	3.18	1.86
2N-with2Neig	0.96	1.38	1.54	1.36	1.18
3N-with2Neig	0.14	0.22	0.35	0.18	0.08
4N-with2Neig	0.04	0.02	0.05	0.02	0.00
5N-with2Neig	0.00	0.01	0.00	0.00	0.00
MaxVisNgb	12.30	12.98	13.32	13.26	15.50
AvgVisNgb	6.22	6.39	6.26	6.24	7.36

2. place all other nodes in *free* set;

3. move all nodes in *free* to *end* set, with new random coordinates and without node overlapping;

4. if node 0 can reach all nodes in *end* set finish the algorithm;

5. move all nodes, in *end* set, not reachable by node 0, to *free* set; and

6. go back to step 3.

We used scenarios with 50, 100, 150, 200 and 250 nodes, positioned in a square with a side of 10 times the radio coverage range. For each number of nodes we generate 50 distinct scenarios and for each scenario we generate 5 versions with shuffled *ids*, in a total of $5 \times 50 \times 5 = 1250$ configurations. Our running scenario is one of these.

Table 3.2 shows some scenarios metrics. The line *With1Neig* contains the number of nodes that only have 1 neighbour, the lines *nN-with2Neig* contain the number of groups of n consecutive nodes with only 2 neighbours (for example, 3N-with2Neig means 3 nodes in line with only 2 neighbours) and lines *MaxVisNgb* and *AvgVisNgb* contain the maximum and average # of neighbours per node, respectively.

Metrics do not present a linear behaviour. That happens because in smaller scenarios some nodes must be regenerated until they are reachable by node 0. This will diverge from randomness and generate node concentration, resulting in a behaviour similar to more crowded scenarios. In scenarios of intermediate size, nodes tend to stay more in the original place and respect a random disposition.

3.2.4.3 DNE, CLB and CNG results

Table 3.3 characterizes the scenarios after Cluster Neighbourhood Gathering (CNG) stage, which includes Dominant Node Election (DNE) and Cluster Building (CLB) stages. The metrics presented are the average: number (#) of clients per GO; number of clusters, i.e. number of GODs; maximum cluster size (cluster members); maximum cluster depth;

Table 3.3: Results for DNE, CLB and CNG stages

	Scenarios size (# of nodes)				
Metric	**50**	**100**	**150**	**200**	**250**
ClientsPerGO	2.18	2.22	2.23	2.26	2.26
Clusters	8.94	16.75	24.71	32.18	34.86
MaxClusterSize	10.83	12.62	13.41	14.00	15.90
MaxClusterDepth	3.53	3.84	3.96	4.05	4.33
GOsRejected	52.55	113.20	167.62	230.48	340.83
LCsToGOsCNG	0.97	2.33	3.55	4.89	5.01
LCsToGOsMCI	0.10	0.26	0.38	0.42	0.65

number of rejected GOs/enslavements; and number of LCsToGOs in CNG and MCI stages.

From these values, we highlight the number of rejected GOs, as it evidences the competition between GOs to capture unconnected neighbour nodes showing that those actions are non-deterministic. Concerning group formation, we did not obtain a high average of clients per GO. We point $L = 5$ as one contribution to that value. Another factor is that, in CLB stage, when a node is enslaved it turns into a GO and also starts enslaving. GOs enslaving concurrently, in CNG stage, shorten the stage time but do not maximize the number of clients per GO.

The number of LCsToGOs in MCI stage is mentioned here to enable direct comparison with LCsToGOs in CNG stage. From these values we conclude that the LCsToGOs mainly occur in CNG stage. However, LCsToGOs of the MCI stage are important to prevent bottlenecks that can occur in that stage, as will be shown in Section 3.2.4.11 (LCsToGOs results).

3.2.4.4 MCI and FCI connectivity results

Here we measure the overall connectivity after the MCI and FCI stages. Figure 3.9 shows the percentage of not fully connected scenarios after those two stages for each used scenario size and also the global average for all scenarios sizes.

When using only MCI stage, we obtain a global result of 7.12%. When using MCI and FCI stages we obtain 2.72%, which is 38.2% of the result when using only MCI stage. Therefore, the addition of a second interconnection round reduced the average of not fully connected scenarios by almost 2/3. The algorithm presented similar behaviour in smaller and larger scenarios, confirming that the generation algorithm should join nodes in sparse scenarios, creating node dispositions like the more crowded ones.

3.2.4.5 Communication results

Now we want to measure the communication cost of each stage. Figure 3.10 contains the average number of broadcasts and unicasts for all scenario sizes and for all algorithm

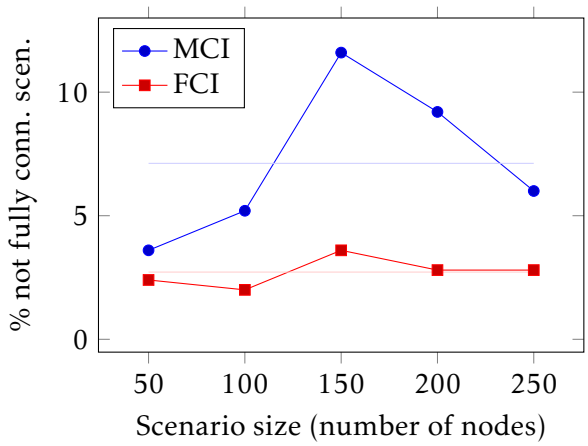


Figure 3.9: Connectivity after MCI and FCI stages.
Horizontal lines show global average values.

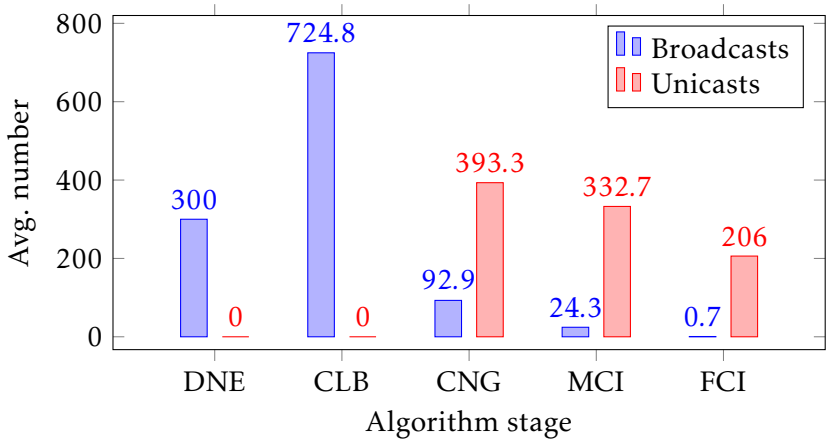


Figure 3.10: Communication results for all stages.

stages.

In DNE stage nodes need 2 broadcasts to get the second neighbourhood. As we have an average of 150 nodes, between all scenarios, the algorithm needs $2 * 150 = 300$ broadcasts. In CLB stage nodes send broadcasts when they change state (to GO, new slave, etc.), reject being enslaved or when they try to enslave nodes and repeat after rejection. In CNG stage every node sends the information of its neighbourhood to their GODs and these ones send information to higher GODs. Thus, communication inside clusters, which goes by unicast, is higher than between clusters, which goes by broadcast. This stage also sets LCsToGOs and for each one of them it requires several unicasts and one broadcast.

The MCI and FCI stages do cluster interconnection that require many unicasts and very few broadcasts. They also send the final end of stage messages to neighbour clusters that also require more unicasts to travel inside and between connected clusters than the broadcasts needed to communicate among unconnected clusters. The MCI stage makes the most of cluster interconnections, so it requires more unicasts and broadcasts than the FCI stage. The global average number of broadcasts and unicasts, for *RedMesh*, is,

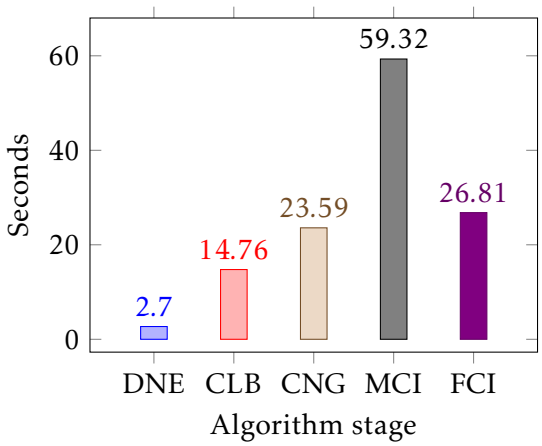


Figure 3.11: Time results for all stages.

respectively, 1142.7 and 932.0. This shows that 45% of the communication can take advantage of unicasts, which are faster and require less energy than broadcasts.

3.2.4.6 Time results

Now we want to measure the time cost of each stage. Figure 3.11 contains the average required time to finish each algorithm stage, for all scenario sizes.

From the figure we can see that MCI stage is the most demanding of all. That occurs because it makes almost every cluster interconnection and that requires: sending information to gateways, communicating between gateways, handling nodes changes (for example, turning into GO), handling rejections and communicating back to GODs. Besides that, GODs have to send final messages to higher clusters. All these actions are done sequentially and several of them require broadcasts, which are slower than unicasts.

The time for CNG and FCI stages is similar as both essentially execute the communication round. From them, FCI stage may have to execute some interconnection rules that increases its time comparatively to CNG stage time.

The average for all scenarios sizes and for the complete algorithm is 127.18 seconds.

3.2.4.7 Cluster interconnection results by type of interconnection

Here we want to know the number of cluster interconnections that occurred in MCI and FCI stages. More precisely, we want to know the number of cluster interconnections done, achieved by intermediation and not done at all.

Figure 3.12 contains the average number and standard deviation of interconnections, separated by its type and for MCI and FCI stages, in scenarios with just 250 nodes. From the figure, we can see that 99.7% of the cluster interconnections done occur in MCI stage. The FCI stage makes very few cluster interconnections, however, it enables the average global final result to improve from 7.12% to 2.72% of not fully connected scenarios, as already seen.

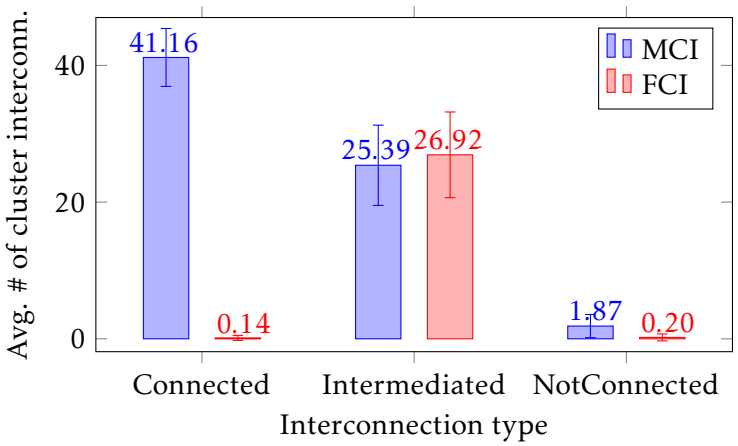


Figure 3.12: Cluster interconnections in MCI and FCI stages.
Average number and standard deviation of cluster interconnections in scenarios with 250 nodes, by type of interconnection.

We can see that both interconnection stages/rounds establish a significant number of clusters interconnections by intermediation. The MCI presents a smaller value than the FCI, as the latter uses all the interconnections by intermediation from the former stage and could add more due to new connections meanwhile established in the both stages. The total number of cluster interconnections by intermediation is even bigger than the total number of cluster interconnections done. That shows that many connections will not be done and the nodes that those connections would otherwise use, will stay free to be used by the critical connections.

The number of unconnected clusters is low, but ideally it should be taken to 0, as pairs of clusters that can not be connected directly or indirectly may create network segmentation.

3.2.4.8 Cluster interconnection results by rule

Here we analyse the cluster interconnections done by each interconnection rule in MCI and FCI stages. The global average number of interconnections done, per scenario, is 25.94 for MCI and 0.13 for FCI, confirming that the latter acts as a complementary stage. However, the FCI stage is very important for the overall connectivity, as already shown in Figure 3.9.

Figure 3.13 contains the average number and the standard deviation, per scenario, of applications done for each rule in MCI stage, in scenarios with 250 nodes. From the figure, it can be seen that the R1 rule is the most significant rule in MCI stage. That is expected, as having pairs of LC and GO gateways, in range, should be a frequent situation. The next significant rule that follows is the R3 rule, mainly because it is the first rule to be checked. The last one to have significant results is the R2 rule that connects pairs of LC gateways in range and complements the R1 rule. The R5 rule solves the cases of pairs of GOs in range, but as it is the last rule to be checked, it is not often used. The R4 rule is

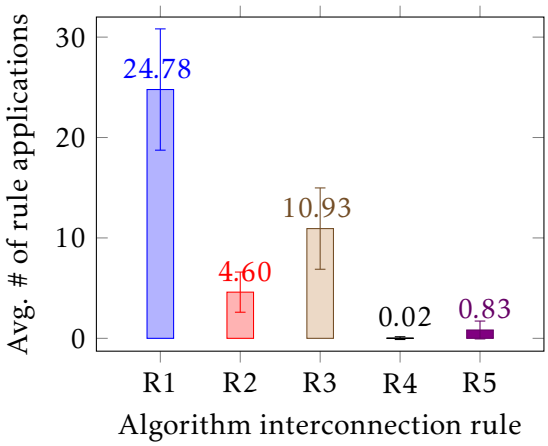


Figure 3.13: MCI interconnection rule applications.
Average number and standard deviation of rule applications in scenarios with 250 nodes.

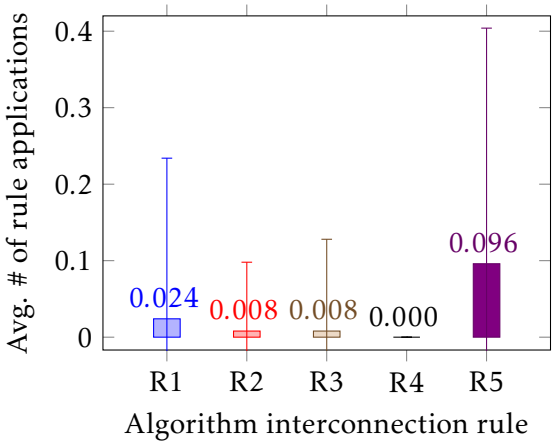


Figure 3.14: FCI interconnection rule applications.
Average number and standard deviation of rule applications in scenarios with 250 nodes.

a complement of the R3 rule and as can be seen, is rarely used.

Figure 3.14 contains the average number and standard deviation, per scenario, of applications done for each rule in FCI stage, in scenarios with 250 nodes. We can see that the R5 rule is the prevailing rule in FCI stage. This is due to the fact that some LCsToGOs are turned again into LCs to enable cluster interconnections, as GOs without any slaves are elected by the R5 rule (without any delegations).

3.2.4.9 Failed cluster interconnection results

Here we analyse the failed cluster interconnections. Figure 3.13 contains the average number and the standard deviation of failed rule applications in MCI stage, per scenario and in scenarios with 250 nodes.

From the figure we can see that R1, R3 and R2 have more rejections than R4 or R5. That occurs because they are the ones with more applications in MCI stage. The values, for R1, R3 and R2 successful applications are 189X, 119X and 115X more than the

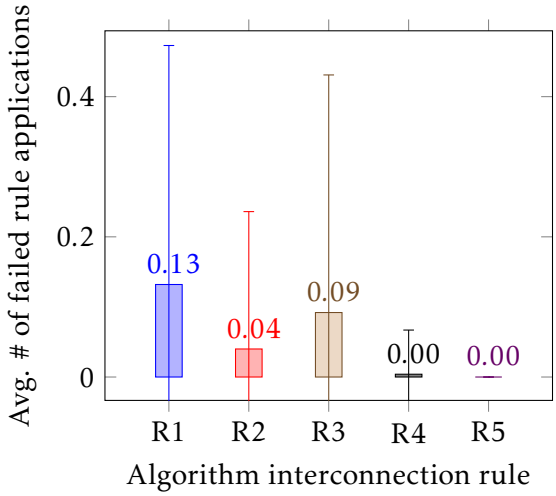


Figure 3.15: MCI failed interconnection rule applications.
Average number and standard deviation of failed rule applications, per scenario, in scenarios with 250 nodes.

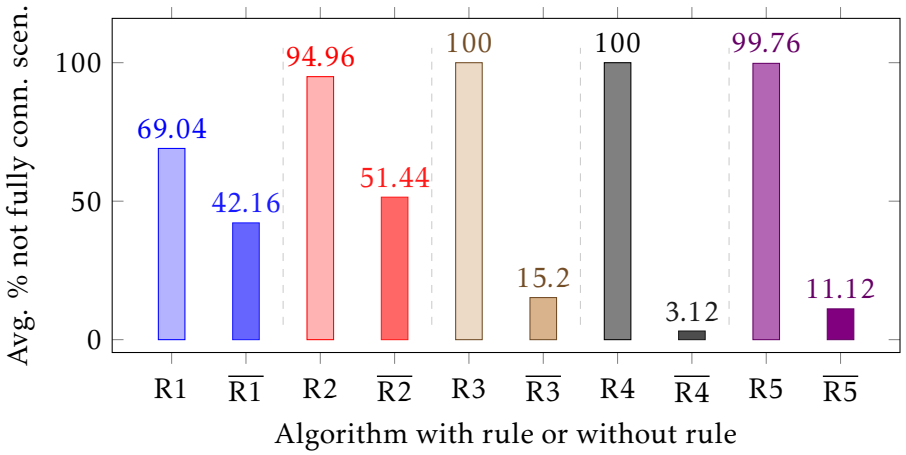


Figure 3.16: *RedMesh* with one rule (Rx) and without one rule ($\overline{\text{Rx}}$).

respective values for the number of failed rule applications. These show that one failed rule application happens in about one hundred rules applications for R3 and R2, and in almost two hundred for R1.

The failed rule applications occur due to conflicts between concurrent clusters, as the changes in gateway nodes, introduced by one cluster, may be unknown by GODs from the neighbour clusters.

3.2.4.10 Interconnection rule relevance results

Here we question the relevance of each interconnection rule by using just one of them and also using all except one. Figure 3.16 shows the global average % of not fully connected scenarios when using only one rule in Rn columns and when using all rules except one in $\overline{\text{Rn}}$ columns (here, the name means: without that rule).

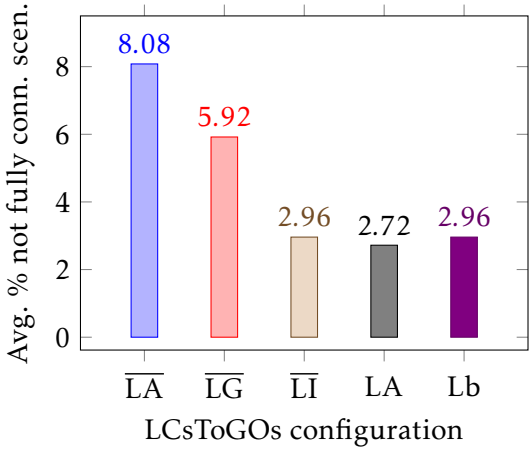


Figure 3.17: LCsToGOs results (LA ≡ *RedMesh*).

From the figure it can be seen that R1 is the only rule that by itself can connect a relevant percentage of scenarios (30.96%), confirming R1 as a very versatile rule. From the columns with all except one, we can observe that R1 and R2 are the most relevant rules, because without these rules the final result of not fully connected scenarios increase significantly. R3, R4 and R5 rules act as a complement, but contribute to the low final percentage of not fully connected scenarios of 2.72%. $\overline{\text{R4}}$, which is *RedMesh* without R4, is an example of that with an average of 3.12% of not fully connected scenarios, showing that R4 makes a very small contribution to the final result (2.72%).

3.2.4.11 LCsToGOs results

Here we want to evaluate the relevance of LCsToGOs.

Figure 3.17 shows the global average percentage of not fully connected scenarios in different settings, namely: $\overline{\text{LA}}$, not using any LCsToGOs; $\overline{\text{LG}}$, not using LCsToGOs in CNG stage; $\overline{\text{LI}}$, not using LCsToGOs in MCI stage; LA, using all LCsToGOs (in CNG and MCI stages, it is the default configuration); and Lb, using all LCsToGOs but they are blocked to remain LCsToGOs in MCI stage.

From the values in Figure 3.17, we can see that LCsToGOs improve the average percentage of not fully connected scenarios, reducing it from 8.08% to 2.72%. From CNG and MCI LCsToGOs, the CNG ones have more impact on algorithm results, as they occur about ten times more than the others, as seen in Table 3.3. The LCsToGOs from MCI stage help with a slight contribution, but solve the situations resulting from connections done, where one gateway is the only possibility to provide connection for three or more clusters.

Initially we conceived the LCsToGOs with the intention to preserve them as GOs in the MCI phase. In that case, only in FCI and only if needed, could they give up from being a GO to make a connection that otherwise would not be possible. We planned it that way, because in FCI is the last chance to make connections and connecting to one of

two or more neighbour clusters is better than not connecting to any of them.

The Lb column of Figure 3.17, shows the results with this behaviour, which is a global average of 2.96% of not fully connected scenarios. The LA column, which is the default *RedMesh* configuration, shows the results when enabling LCsToGOs to give up from being GOs in MCI and achieved 2.72%. In this case (LA), the LCsToGOs only act to prevent higher clusters from transforming them in CR (LC/CL) nodes.

Consequently, we by default, in *RedMesh*, allow LCsToGOs to be selected by the R5 rule (without any delegations) and give up from being GOs in any interconnection stage.

3.2.4.12 Interconnection rule R2 results

Here we want to analyse the behaviour of the cluster interconnection rule R2. Concretely, we want to know if the used version is the one that maximizes connectivity. *RedMesh* only uses one version of R2, but as here we will discuss two versions we call the used version R2a. The R2a turns the local gateway into GO and the remote gateway connects its WFD interface to the local one. This version has a global average percentage of not fully connected scenarios of 2.72%, as presented in several figures, like in Figure 3.17 (column LA).

The R2b version turns the remote gateway into GO and the local gateway connects its WFD interface to the remote one. The same kind of result for the application of this version R2b is 2.8%.

We expected that R2b would be more effective, as we thought that having the GO in the remote cluster would give more freedom of connectivity than having it in the local cluster. However, the results do not confirm that.

Therefore, we use R2a as the default version for *RedMesh*.

3.2.4.13 Rule order results

Here we want to evaluate the impact of the order of the application of the rules. We use a notation where rules are tested following the appearance order of their numeric value after 'R', e.g. "R31245" means rules are applied by order R3, R1, R2, R4 and R5.

Figure 3.18 contains the results for the following rule order configurations (short and figure column name in parentheses): default algorithm order, R31245 (O3); no priority to R3, R12345 (O1); priority to R2 over R1, R32145 (O2); priority to R4 over R1 and R2, R34125 (O4); and priority to R5, R53124 (O5).

Results show that R3 should be tested before R1 and R2, R2 should be after R1, R4 should be after R1 and R2, and that R5 should not be tested first.

Thus, we empirically conclude that R31245 (O3), which is the *RedMesh* order, is the best.

3.2.4.14 Results from cluster connections with/without intermediation

Here we will evaluate the impact of having or not cluster connections by intermediation.

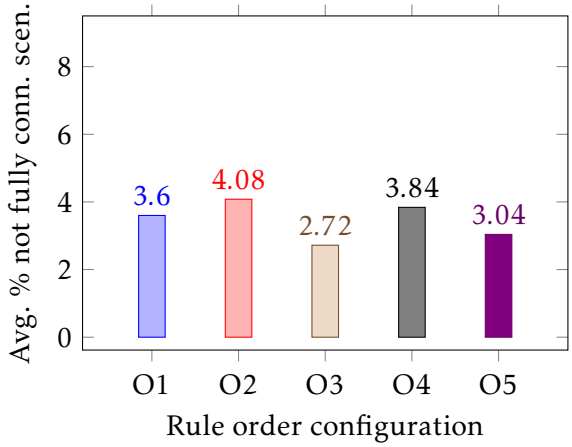


Figure 3.18: Rule order results (O3 ≡ *RedMesh*).

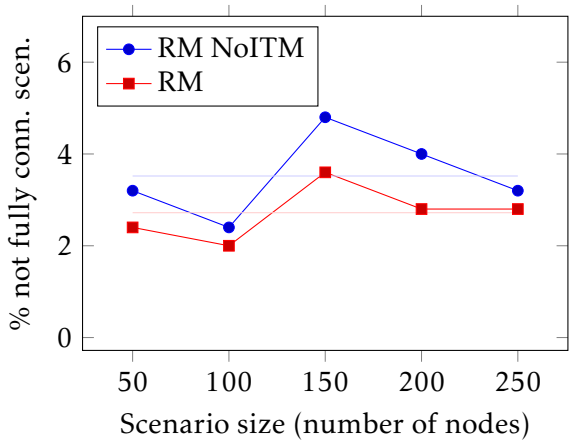


Figure 3.19: Connectivity with and without intermediate connections.

For that purpose we set a test that does not use any kind of intermediate cluster connection. Hence, when there is not a direct connection, the algorithm just says that there is no connection possible. With this setup, we obtain a global average of 3.52% of not fully connected scenarios. This value is quite close to the value when intermediate connections are used, which is 2.72%.

Figure 3.19 contains the results for each group of scenarios and the global averages for algorithm with and without using the intermediate cluster connections, respectively in RM values (red squares) and in RM NoITM values (blue circles). From the figure we can see that it was in scenarios with 150 and 200 nodes where the absence of intermediate cluster connections had more impact. We point out that, the reason for that behaviour, should be our scenario formation algorithm. We affirm that because this algorithm tends to form scenarios with some similar properties between scenarios with few nodes (50) and with many nodes (250).

The RM NoITM version presented, comparatively with RM and relative to the global averages, more 2.0% of time, more 1.1% of broadcasts, less 5.4% of unicasts and more

37.0% and 348.3% (almost 3.5X) of connections in MCI and FCI, respectively. We can see that more cluster connections take slightly more time and more broadcasts, but with those connections the networks reduce the number of unicasts, as they do not have the long intermediate paths composed mainly of unicasts.

Nevertheless, as RM has less 23% of not fully connected scenarios than RM NoITM, we conclude that using intermediate connections improves the overall connectivity.

3.2.4.15 L (maximum number of slaves) results

Here we will evaluate the impact of changing the maximum number of slaves per GO. We always used $L = 5$ as that value is the most restrictive that can be used. Here we test the other values that can be used: 6, 7 and 8.

The experiments with $L \in [6,7,8]$ gave the same results, with a global average (for scenarios of all sizes) of 2.80% of not fully connected scenarios. The similar value for $L = 5$ (the default value) is 2.72% (as seen several times). As results are equal for $L \in [6,7,8]$, from now on we will only mention $L = 6$.

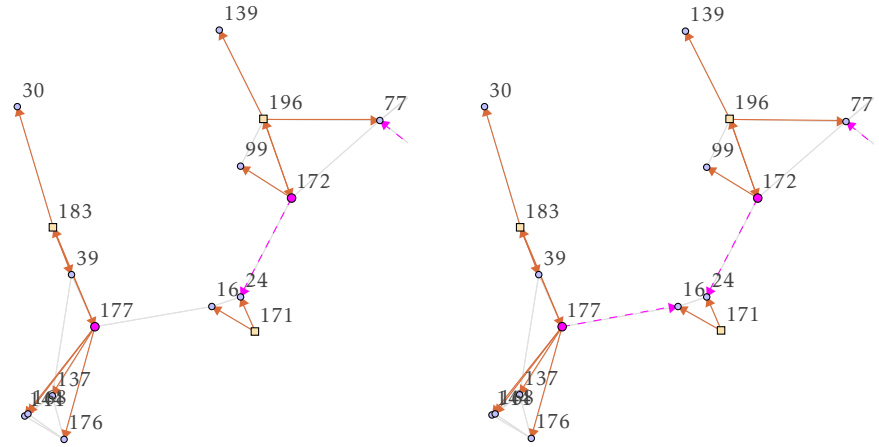
We observed that with $L = 5$ there are 34 not fully connected scenarios and with $L = 6$ there are 35. Out of these 35, 33 scenarios have common results between $L = 5$ and $L = 6$.

Only in one scenario, $L = 5$ did not achieve connection and $L = 6$ did. That scenario has 200 nodes and is shown in Figure 3.20 (only the relevant part of it). The figure presents the results when using $L = 5$ on the left and when using $L = 6$ on the right. We can see that when using $L = 5$, the GO 177 has 5 slaves (GOD 183 by WiFi and 4 LCs: 137, 176, 144 and 168) and its WiFi interface is connected to GOD 183. Therefore, this node cannot connect to, or be connected by, any other node.

With $L = 6$, the extra slave possibility enabled to solve this situation. Therefore, GO 177 received node 16 as a slave connected by its WFD interface.

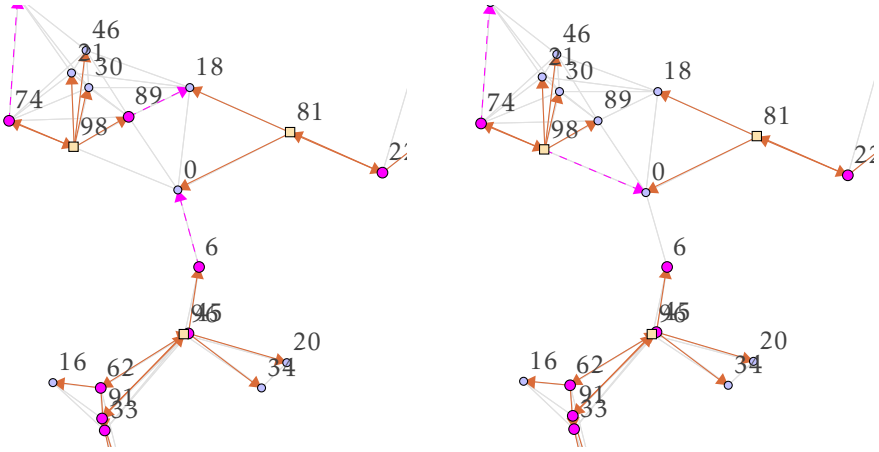
To enable connection with $L = 5$, we will need a *new interconnection rule with the pre-conditions of having: a local LC/GO node with L slaves and that can delegate one slave; and a remote LC node*. The rule actions are firstly to execute the delegation and secondly to ask the remote LC to connect its WFD interface to the local GO. Due to concurrent actions in remote clusters, we do not use delegation in remote clusters. That is why R5 interconnection rule does not have a mirrored version (R5b). Therefore, this newly proposed delegation rule will not have a mirrored version either (with a remote GO, which would delegate one node, and local a LC). This new rule would then delegate one slave from GO 177 to a local node, such us delegate LC 176 to LC 137 (turning this node into a GO first), and the remote node, the LC 16, would connect its WFD interface to the local GO (177).

In Figure 3.21 we show the first scenario where $L = 5$ achieves full connection and $L = 6$ does not. This scenario has 100 nodes and presents a connection problem around node 0, which is a node local to cluster 81 (cluster with GOD 81). Node 0 is a candidate gateway also in clusters 98 and 96. Clusters 98 and 96 are not neighbour clusters, as they



a: Scenario with $L = 5$ (not conn.). b: Scenario with $L = 6$ (conn.).

Figure 3.20: Scenario not connected with $L = 5$ and connected with $L = 6$.
Solid brown and dashed pink arrows are, respectively, WiFi and WFD (master → slave) connections (connected from slave to master). Yellow squares, larger pink circles and smaller blue circles depict, respectively, dominant nodes (GODs), LC/GO nodes and LC nodes. Grey lines are the radio visibility.

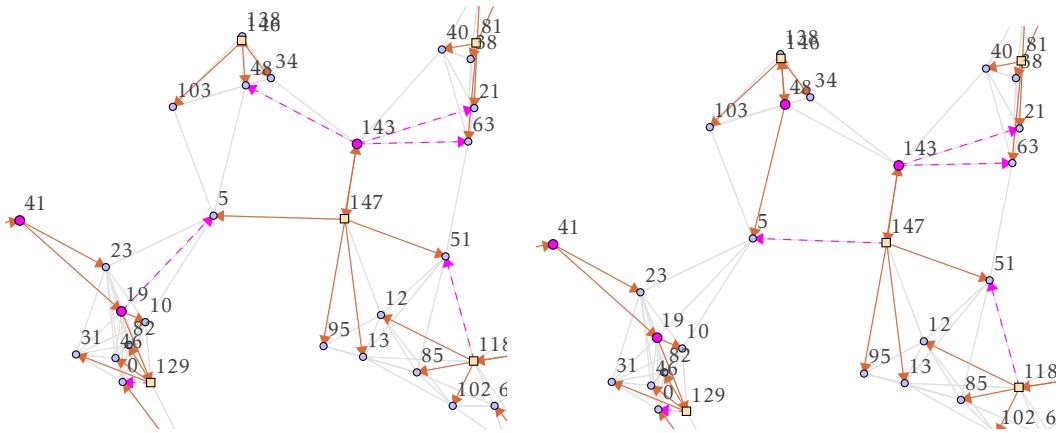


a: 1^{st} scenario with $L = 5$ (conn.). b: 1^{st} scenario with $L = 6$ (not conn.).

Figure 3.21: 1^{st} scenario connected with $L = 5$ and not connected with $L = 6$.
Solid brown and dashed pink arrows are, respectively, WiFi and WFD (master → slave) connections (connected from slave to master). Yellow squares, larger pink circles and smaller blue circles depict, respectively, dominant nodes (GODs), LC/GO nodes and LC nodes. Grey lines are the radio visibility.

do not have a common candidate gateway. Thus, they execute MCI stage independently. However, from simulations we observe that GOD 98 runs MCI stage before GOD 96. GOD 81 did not set node 0 as a LCToGO in CLB stage. This happens because for GOD 81, the cluster 98 can be connected to node 18 and node 0 only needs to connect to cluster 96. Consequently, for GOD 81, node 0 should not be a LCToGO node.

Figure 3.21a shows the results for $L = 5$. We can see that GOD 98 cannot enslave node 0, as it already has L (5) slaves (89, 74, 46, 30, 21), so it used the R2 rule to connect to



a: 2^{nd} scenario with $L = 5$ (conn.). b: 2^{nd} scenario with $L = 6$ (not conn.).

Figure 3.22: 2^{nd} scenario connected with $L = 5$ and not connected with $L = 6$. Solid brown and dashed pink arrows are, respectively, WiFi and WFD (master → slave) connections (connected from slave to master). Yellow squares, larger pink circles and smaller blue circles depict, respectively, dominant nodes (GODs), LC/GO nodes and LC nodes. Grey lines are the radio visibility.

cluster 81. That rule turned node 89 into a LC/GO and then node 18 connected its WFD interface to node 89. Then GOD 96, which is the next GOD to run MCI stage, connected to cluster 81, using the R2 rule between nodes 6 and 0. Therefore, all three clusters, 98, 96 and 81 were connected and the scenario achieved full connection.

Figure 3.21b shows the results for $L = 6$. Now, GOD 98 has the possibility to enslave one more node. Then, the R1b rule selected GOD 98 and node 0. This blocked any possibility to connect to cluster 96, creating a network with two separate segments.

In general terms, the problem is that R1b rule only considered the pair of nodes 98 and 0, while R2, which is another possible rule, would select the pair of nodes 89 and 18, where the remote node is a candidate gateway to connect less networks than the first pair. However, as the Cluster Interconnection Procedure (CIP) only compares pairs on the same rule version and not pairs of distinct rules, the second pair was not selected. Thus, one solution for these cases is to *change CIP, to first check all the rules and their versions and second to select only the most favourable pair* among all the possible ones. This way, the procedure will return a global decision, among rules and their versions, and not the first one to be found.

In Figure 3.22 we show the second scenario where $L = 5$ achieves full connection and $L = 6$ does not. This scenario has 150 nodes and presents a connection problem around node 5 and among clusters 147, 146 and 119 (not visible in the figure, but it is the GOD of nodes 41, 23, 19 and 10).

When using $L = 5$, in Figure 3.22a, node 5 is local to cluster 147. As GO 147 is the first one to run the MCI stage, it connects firstly to cluster 146, connecting nodes 143 and 48 and secondly to cluster 119 connecting nodes 5 and 19. So, these clusters were connected and the algorithm achieved full connectivity in the scenario.

When using $L = 6$, in Figure 3.22b, node 5 is local to cluster 146. That happened due to concurrency in CLB stage and shows that if we change one parameter, the algorithm can present distinct situations. Now the GOD 147 has node 143 as an LC node that is a gateway to cluster 146, 88 (not visible) and 81 and node 5 that is a gateway to cluster 119 (not visible). Therefore, GOD 147 started by trying to connect to cluster 146, which is the highest from its lower neighbour clusters. Then the pair of nodes 147 and 5 is selected, because node 5 can connect to one cluster (119) while node 143 can connect to two clusters (88 and 81). GOD 147 did the following actions: connected to cluster 146 using R1b and nodes 5 and 147; tried to connect to cluster 119, but failed; connected to cluster 118 using R1a and nodes 51 and 118; turned node 143 into a LCToGO; connected to cluster 88 using R1b and nodes 147 and 21; and finally connected to cluster 81 using again R1b and nodes 147 and 63.

We conclude that the problem here was almost the same as in the previous scenario, as there is one node that is the only node to connect to one cluster and there is another cluster that can be connected by that node and by another one. The common gateway is not set as a LCToGO because it is not the only node to connect to two or more clusters. Hence, it may be used in other interconnections and as a CR node and definitively block the connection to the cluster, as it is the only node that can provide connection. Here we can see that GOD 147 has to connect to clusters 146, 119, 118, 88 and 81 and the best solution is the one that maximizes current and future connections between all these clusters and neighbour clusters of all nodes in all these clusters. Not choosing to go for a global solution, we identify a local one, which consists of *enlarging the scope of LCToGO nodes* to be any LC node that is the unique local node to connect to one cluster and is a candidate gateway to connect to another cluster as well. This definition covers the current LCsToGOs scope, because it includes the cases of nodes that are the unique local nodes to connect to two or more clusters.

We conclude that when using $L \in [6, 7, 8]$ the results are not much different than when using $L = 5$. We affirm that, because the difference between values is short and because when we change one parameter in the algorithm, different situations appear and node connectivity depends mainly on the rules used. Nevertheless, we expect to get better results when using larger values of L.

With the analysed situations, we identified some weaknesses in our algorithm and pinpointed possible solutions to improve results.

3.2.4.16 Comparison with existing algorithms

Here we evaluate our algorithm against the existing ones: MAGNET [5] and Smart-Group-Formation (SGF) [9].

The MAGNET algorithm [5] only uses CR–GO connections and does neither mention any kind of delegations (like our R5 rule), nor any kind of node reservations (like our LCsToGOs), or the execution of a second interconnection round (like our FCI stage).

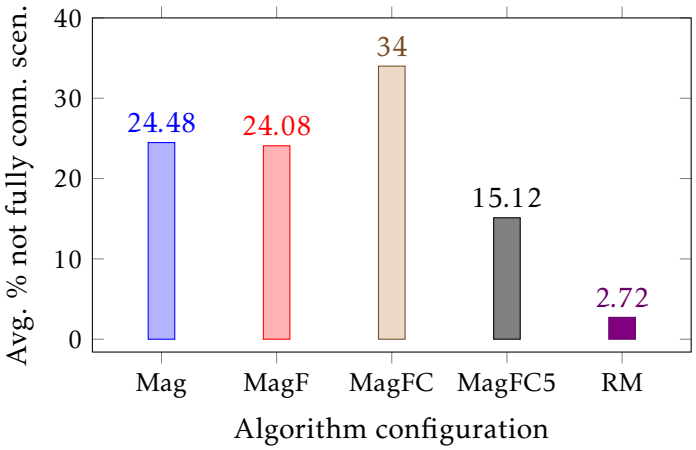


Figure 3.23: *RedMesh* (RM) versus MAGNET (Mag).

Therefore, we emulate this algorithm using only R1 and R2 rules and do not use the FCI stage or LCsToGOs. We call this setup: Mag.

Figure 3.23 contains the global average percentage of not fully connected scenarios, for Mag (on the left bar), some improved versions of Mag and our algorithm *RedMesh* in RM (on the right bar).

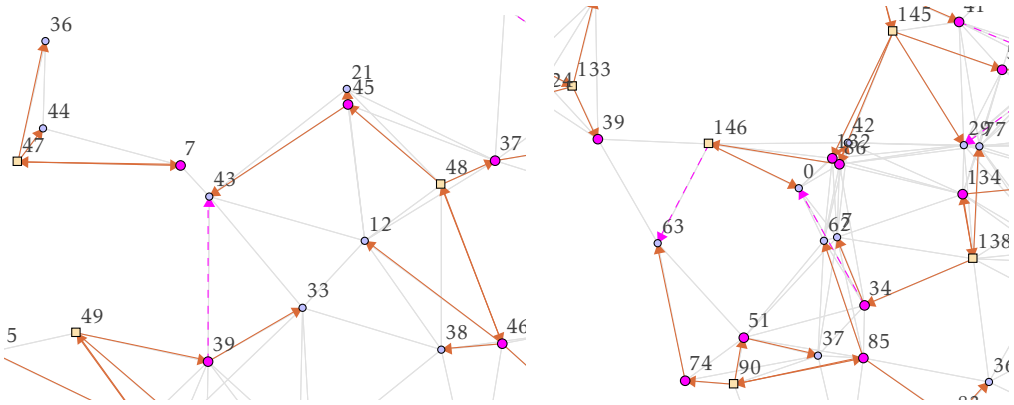
Results show that Mag has 9X more not fully connected scenarios than RM. If we add FCI to Mag (we call it MagF), the value reduces to 8.85X. If we add LCsToGOs to MagF (we call it MagFC), we obtain worse results. That happens because LCsToGOs will not give up from being GOs and will block gateway nodes that are GOs. If we add R5 to MagFC, (we call it MagFC5), we unblock LCsToGOs nodes and obtain a value of 5.56X worse than RM. Thus, after improving the basic version (Mag) with all our strategies we conclude that MAGNET should present worse connection values than *RedMesh*. Nevertheless, MAGNET uses groups (clusters of depth 1, which are clusters formed only by a GO and its slaves), while *RedMesh* uses clusters of variable depth. This aspect should not change the presented results, as having clusters of depth 1 means that GODs are usually closer to neighbour clusters, than when using clusters with any depth, but that proximity will only favour the use of rules R3 and R4, and neither MAGNET nor MagFC5 use them.

The SGF [9] algorithm uses direct GO–GO connections, thus, it can only use R3 and R4 rules. With only those rules we did not obtain a single fully connected scenario. SGF uses clusters of depth 1, but that factor should not be enough for a significant change. We, then, estimate that SGF should present much worse connectivity than *RedMesh*.

3.2.5 Algorithm limitations

Here we want to analyse the algorithm limitation to achieve 100% of fully connected scenarios.

In Section 3.2.4.15 we already identified the following three new features that can improve the algorithm:



a: Scenario 1. b: Scenario 2.

Figure 3.24: Scenarios not connected by *RedMesh*: 1 and 2.
Solid brown and dashed pink arrows are, respectively, WiFi and WFD (master → slave) connections (connected from slave to master). Yellow squares, larger pink circles and smaller blue circles depict, respectively, dominant nodes (GODs), LC/GO nodes and LC nodes. Grey lines are the radio visibility.

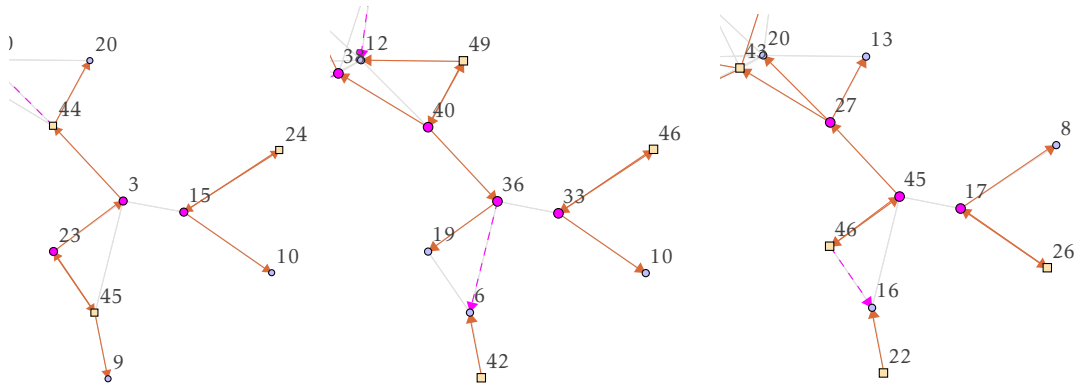
1. a LCsToGOs requirement that selects the nodes which are unique possibilities to connect to one cluster and may be used to connect to one or more other clusters;

2. a change in CIP to consider all rules and choose the best pair from all possible ones; and

3. a new interconnection rule that requires a local LC/GO, which can delegate one slave, and a remote LC.

From the not fully connected scenarios, with the default *RedMesh* parametrization, we found that some of them can be solved using the three identified improvements. The Figures 3.24a and 3.24b are two of them, as they can be connected with the new LCsToGOs requirement. In Figure 3.24a scenario, node 43 is the only node, from cluster 48, which can connect to cluster 47 and is also a candidate gateway to connect to cluster 49. So, it will be a new LCToGO node from cluster 48. If so, cluster 49 will connect to cluster 48 preserving 43 as a GO. Then nodes 43 and 7 will be interconnected by rule R5 from cluster 48 to cluster 47 in MCI or vice versa in FCI. In Figure 3.24b scenario, node 63 is the only node, from cluster 90, which can connect to cluster 133 and is also a candidate gateway to connect to cluster 146. So it will be a new LCToGO node from cluster 90. If so, cluster 146 will not be able to connect to cluster 133. Then it will connect to clusters 138 and 90 but preserving 63 as a GO. Then cluster 133 will connect to cluster 90, interconnecting node 39 and 63, using R1b.

Yet, in other scenarios, the solution to achieve connectivity could be more complex. Figures 3.25a, 3.25b and 3.25c, contain three of these scenarios, which came from the same base scenario but have shuffled identifiers. They all present GO–GO cases that the algorithm cannot connect. In those cases, we proposed a new interconnection rule/action,



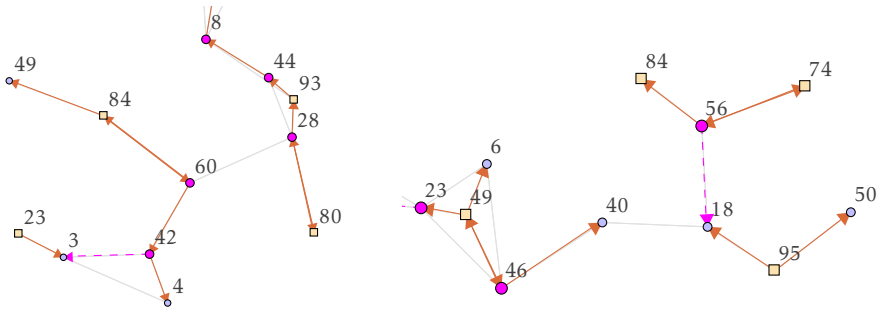
a: Scenario 3. b: Scenario 4. c: Scenario 5.

Figure 3.25: Scenarios not connected by *RedMesh*: 3, 4 and 5.
Solid brown and dashed pink arrows are, respectively, WiFi and WFD (master → slave) connections (connected from slave to master). Yellow squares, larger pink circles and smaller blue circles depict, respectively, dominant nodes (GODs), LC/GO nodes and LC nodes. Grey lines are the radio visibility.

which we call *node capture*. This rule requires a local and a remote LC/GO gateway nodes and the local one cannot delegate its slaves but has availability for one more slave. The rule then proposes to the remote gateway to leave its cluster and join the local cluster by being a slave, by WiFi, of the local gateway. The remote gateway, by receiving such request, should send it to its GOD. Then, the remote GOD will then decide if that node can be captured by the other cluster or not. It will answer positively if the cluster (the remote one) can be connected to the so called local cluster.

Figure 3.26a contains a complete different scenario, but with also a GO–GO unconnected case, which can be solved by this new action. In this scenario, cluster 84 will propose to cluster 80 to capture node 28. GOD 80 will answer affirmatively, because it can connect to node 28, by connecting its WiFi interface to that node. Therefore, node 28 will start by disconnecting from 80, and then connecting its WiFi interface to node 60 (becoming its slave). Then node 80 will connect its WiFi interface to node 28.

Figure 3.26b contains a LC–CR unconnected case. In this scenario, node 18 was elected to be a LCToGO node, but in FCI stage cluster 95 connected to cluster 74, using R5 rule with nodes 18 and 56. In this case, cluster 95 should maintain node 18 as a GO to enable connection to neighbour clusters 74 and 49. To keep node 18 as a GO, we should insist on keeping the LCsToGOs without giving up from being GO in MCI stage. In this scenario, node 18 as a GO can receive the WFD interface from node 40 and then cluster 95 may capture node 56 from cluster 74, as node 74 can keep its WiFi interface connected to node 56. However, tests with LCsToGOs without giving up from being GO in MCI stage presented a worse value than when allowing LCsToGOs to give up from being GO. Thus, we do not have a clear solution for these types of cases, but maybe with the new improvements keeping the LCsToGOs as GOs in MCI stage, the algorithm could give better results than allowing those nodes to give up from being GO.



a: Scenario 6. b: Scenario 7.

Figure 3.26: Scenarios not connected by *RedMesh*: 6 and 7.
Solid brown and dashed pink arrows are, respectively, WiFi and WFD (master → slave) connections (connected from slave to master). Yellow squares, larger pink circles and smaller blue circles depict, respectively, dominant nodes (GODs), LC/GO nodes and LC nodes. Grey lines are the radio visibility.

Consequently, from this section we identify a new behaviour, which we call *node capturing*. We also suggest repeating all the tests with the new identified behaviours.

3.2.6 Reflections about network redundancy and node churn

Here we want to discuss two possible improvements to *RedMesh*: *network redundancy* and *tolerance to node churn*. This discussion aims to contribute to the first considerations for future algorithm improvements in relation to these topics.

Network redundancy

RedMesh was conceived to maximize global connectivity where all nodes in a scenario should be connected in a single network. To achieve that goal, we reduced the cluster interconnection to the identified minimum, so as to maximize the possibility of necessary connections. Therefore, the built networks may have little or no redundancy. In our running scenario, the built network did not present a single redundant cluster interconnection, as shown in Figures 3.7 and 3.8.

If the built networks do not have any redundancy, they are just like a tree of clusters. Therefore, as *RedMesh* goal is to form mesh networks, it needs to add as many cluster interconnections as possible, besides the ones that are currently generated. Thus, strategies are needed to add cluster interconnections without reducing the overall connectivity.

One possible cluster interconnection strategy is to use a third interconnection round. It would be a descending round, where each GOD would connect to all its lower unconnected neighbour clusters, even if they could be connected through intermediation. That strategy is a conservative one, as it ensures that every actual cluster interconnection would exist and only then new ones would be added. This third round would increase the algorithm communication and time. Nevertheless, the network would be ready to work

at the end of FCI stage. Hence, this third round would just improve the network and thus its higher price would just be the energy for its communications.

It would be very interesting to explore strategies to increase connectivity executed inside MCI and FCI stages, but we will leave it for future work.

With such strategies, *RedMesh* would build networks with cluster interconnection redundancy, which will have the benefits already presented in Section 2.2.2 (Topologies analysis), like enabling shorter paths between nodes, enabling traffic splitting and enabling to choose paths with nodes with more battery or with less traffic.

Tolerance to node churn

In relation to node churn, we consider three axes: network state (in formation and formed), node action (node ingress and egress) and type of node (GOD, GO, gateway and LC). We exclude the axes: node positioning, static versus dynamic (movement); and node *identifiers*, immutable versus mutable (with changes over time). We are leaving those axes out of our reflection in this thesis and only considering immediate actions of a node (that can belong to the network).

In relation to *node ingress*, the algorithm must do two steps: node detection and node integration. A new node will start by broadcasting its existence using the BAS and listening for peers. A new node detection may occur before and after clusters are formed. When a new node gets close to a node that does not have a GOD, it will start the algorithm and the neighbour node should collaborate sending the neighbourhood information to it. From then, nodes continue with the algorithm and any node can be a GOD. When the new node detects one node that is already part of a cluster (with a GOD in its BAS information), it will send a broadcast BAS informing the node that it wants to join that cluster. If the new node simultaneously detects more than one cluster, in its neighbourhood, it will choose to join the cluster with higher *id*. When a node from a cluster detects a new node, it sends that information to its GOD. The GOD will decide how the cluster should integrate the new node, choosing the local node that will capture the new node and communicating the necessary actions to the cluster members involved. We will choose to integrate the new node in the existing cluster, independently of the new node *id* (it can be higher than the *id* of the GOD). The GOD will then choose the timing to execute those actions in a way to avoid any occurring cluster changes and afterwards it will register the new node and its neighbourhood information. After the integration of that information, the node will be a normal cluster member available for any cluster operation. These steps are also valid when the algorithm finishes and the network is formed. Therefore they belong to an extended version of the algorithm, which will do the network formation and maintenance.

In relation to node egress, the algorithm must do two steps: node detection and adjustment from node absence. To detect that a node left the network, because it was turned off or its battery ran out, nodes should have a node absence time-out. Node

absence may occur before and after clusters are formed. If that happens before clusters are formed, the nodes in the neighbourhood, after the node absence time-out, adjust their neighbourhood information and the algorithm proceeds with the new conditions. This is valid whether the absent node was supposed to be a GOD or not. If a node absence is detected when the cluster is already built, by default that information is sent to the local GOD and it will coordinate the cluster reconstruction. The GOD may cancel any occurring cluster activity and must repair the cluster immediately. If the absent node is the GOD or a part of the cluster gets separated from its GOD, then the remaining cluster member should elect a new GOD from existing cluster members. These steps are also valid when the algorithm is finished and the network is formed.

In relation to the ingress or egress of a node that is a gateway or can be a gateway for cluster interconnection among neighbour clusters: the clusters should run the MCI and FCI stages between them.

We conclude that the algorithm may present some tolerance to node ingress and egress, in formation and maintenance phases, but it has to be modified/extended to handle these situations.

3.2.7 Conclusion

In this section, we proposed an algorithm, called *RedMesh*, which forms mesh networks of WFD enabled devices. The algorithm uses two efficient connection and communication topologies, allowing formed networks to resort only to unicasts. We confirmed that the selected algorithm aspects, such as rules, rule order, LCsToGOs and two interconnection rounds, are relevant for the overall performance.

Results show that the second cluster interconnection round (FCI stage) improves connectivity, by reducing the cases of not fully connected scenarios in 61.8%, but requires more 28.4% of unicasts and more 26.7% of time. These values ask for improvements like skipping the FCI stage whenever possible or overlapping the FCI and MCI stages to reduce total time.

The algorithm performed better than emulated versions of existing algorithms for scenarios up to 250 nodes. The algorithm characteristics do not show any evidence that can inhibit its use in larger scenarios.

Finally, we conclude that *"it is possible to systematically form WFD mesh networks in large scale scenarios of WFD enabled devices"*.

3.3 Conclusions

In this chapter we first identified the existing WFD network formation algorithms and their drawbacks to be used in large-scale scenarios. Then we proposed *RedMesh*, an algorithm that uses five interconnection rules, three communication rounds, the GOCRGO

and GOGO topologies and among connected nodes it can communicate by unicasts (either in algorithm execution and in future network use). This algorithm achieved 97.28% of fully connected scenarios in a total of 1250 test cases with up to 250 nodes. So, we affirmatively conclude that *"It is possible to systematically form WFD large scale mesh networks"*.

Chapter 4

Conclusion

The main research question of this thesis is: MQ – *"Is it possible to build large scale mesh networks of WFD enabled devices?"*. That question was split in two base questions: Q1 – *"Is it possible to do WFD intergroup connection and communication for large scale mesh networks?"* and Q2 – *"Is it possible to systematically form WFD large scale mesh networks?"*.

We will now present the final conclusions about the base questions and then about the main question. Afterwards we will discuss future work and open issues.

4.1 Q1 – WFD intergroup for large scale mesh networks

We start the first base question (Q1 – *"Is it possible to do WFD intergroup connection and communication for large scale mesh networks?"*) by validating it as a research question and then solving it by presenting three connection and communication topologies. We only identified one existing topology (GOCR) which requires the use of broadcasts, many nodes and transmission per WFD group and is tailored only for tree like structures. We then proposed three topologies (GO2CR, GOCRGO and GOGO) to provide ways to interconnect WFD groups to create mesh and tree like structures, using unicast communication and requiring the minimum number of nodes and transmissions per WFD group.

Besides that, we developed a variant ($GOCR_{UC}$) of the existing topology (GOCR), which can be used by devices with WFD as *priority interface* and only requires the use of unicast communication.

We also analysed and evaluated all the topologies and concluded that our developed topologies are better than the existing topology, in almost every aspect, being the most relevant: the capability to build mesh (and tree) like structures, the communication speed and the number of used nodes. Therefore, the developed topologies enable us to conclude,

now, that *"it is possible to do WFD intergroup connection and communication for large scale mesh networks"*.

4.2 Q2 – WFD large scale mesh network formation

The second base question (Q2 – *"Is it possible to systematically form WFD large scale mesh networks?"*) was first validated as a research question. We concluded that existing WFD network formation algorithms (SGF and MAGNET), meanwhile developed, are not appropriate to create mesh networks (SGF) or offer limited support for mesh networks (MAGNET). These algorithms also use communication topologies that require UDP broadcasts, that are slow and energetically demanding, presenting a strong drawback for their use.

Then, we presented our developed algorithm, *RedMesh*, which is tailored to build mesh networks in a decentralized way and to allow the use of UDP unicast communication in its execution and in the networks built. The algorithm is characterized by having three communication rounds, five interconnection rules and using gateway nodes in a way to maximize cluster interconnections. The algorithm achieved connectivity in 97.28% of the 1250 scenarios, with up to 250 nodes and performed better than the emulated version of existing algorithms.

Therefore, with the *RedMesh* algorithm, we conclude that, now, *"it is possible to systematically form WFD large scale mesh networks"*.

4.3 MQ – WFD large scale mesh networks

With the topologies and algorithm developed, we affirmatively conclude that, now, *"it is possible to build large scale mesh networks of WFD enabled devices"*.

Consequently, the way for the use of WFD in medium and large scale scenarios and for autonomous collaborative mobile systems is now paved. Nevertheless, the lack of an efficient use of both interfaces (WiFi and WFD) simultaneously is a major drawback for an effective use of WFD in these scenarios.

4.4 Future work and open issues

As future research work, we foresee to extend the network formation algorithm and to improve the comparison with other algorithms.

We intend to extend the *RedMesh* network formation algorithm to:

- improve cluster interconnection (with the identified new features),
- make redundant cluster interconnections to improve node shortest average path and network redundancy, and
- have tolerance to node churn.

We envision to improve the comparison of *RedMesh* with other algorithms and the *RedMesh* evaluation by:

- comparing it with an algorithm that forms only tree networks and
- evaluating it in a full communication stack simulator.

There are also the following issues that are of much interest and are open research questions:

- Can the algorithm features introduced in *RedMesh* improve BSF algorithms?
- How WiFi-Aware and MeshTalk will behave, in relation to this work, in medium and large scenarios?

Bibliography

[1] W. Alliance. *Neighbor Awareness Networking Technical Specification*. Version 2.0. accessed: 2019-06-27. 2017. URL: https://www.wi-fi.org/file/neighbor-awareness-networking-specification.

[2] A. Asadi and V. Mancuso. "WiFi Direct and LTE D2D in action." In: *IFIP Wireless Days Conference*. WD'13. Valencia, Spain: IEEE, Nov. 2013, pp. 1–8. DOI: 10.1109/WD.2013.6686520.

[3] M. Baert, J. Rossey, A. Shahid, and J. Hoebeke. "The Bluetooth Mesh Standard: An Overview and Experimental Evaluation." In: *Sensors* 18 (July 2018), p. 2409. DOI: 10.3390/s18082409.

[4] L. Baresi, N. Derakhshan, and S. Guinea. "WiDiSi: A Wi-Fi direct simulator." In: *IEEE Wireless Communications and Networking Conf.* WCNC'16. Doha, Qatar, Apr. 2016, pp. 1–7. DOI: 10.1109/WCNC.2016.7565169.

[5] L. Baresi, N. Derakhshan, S. Guinea, and F. Arenella. "MAGNET: A Middleware for the Proximal Interaction of Devices based on Wi-Fi Direct." In: *Int. Conf. on Communications*. ICC'17. Paris, France: IEEE, May 2017, pp. 1–7. DOI: 10.1109/ICC.2017.7997222.

[6] Bluetooth SIG. *Bluetooth specifications @ONLINE*. Accessed: 2018-01-08. URL: www.bluetooth.com/specifications.

[7] U. Botrel Menegato, L. Souza Cimino, S. E. Delabrida Silva, F. A. Medeiros Silva, J. Castro Lima, and R. A. R. Oliveira. "Dynamic Clustering in WiFi Direct Technology." In: *12th ACM Int. Symp. on Mobility Management and Wireless Access*. MobiWac'14. Montreal, QC, Canada: ACM, 2014, pp. 25–29. ISBN: 978-1-4503-3026-8. DOI: 10.1145/2642668.2642682.

[8] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano. "Device-to-device communications with Wi-Fi Direct: overview and experimentation." In: *IEEE Wireless Communications* 20.3 (July 2013), pp. 96–104. ISSN: 1536-1284. DOI: 10.1109/MWC.2013.6549288.

[9] C. Casetti, C. Chiasserini, Y. Duan, P. Giaccone, and A. Perez Manriquez. "Data Connectivity and Smart Group Formation in Wi-Fi Direct Multi-Group Networks." In: *IEEE Transactions on Network and Service Management* 15.1 (Mar. 2018), pp. 245–259. ISSN: 1932-4537. DOI: 10.1109/TNSM.2017.2766124.

[10] C. E. Casetti, C.-F. Chiasserini, L. C. Pelle, C. D. Valle, Y. Duan, and P. Giaccone. "Content-centric routing in Wi-Fi Direct multi-group networks." In: *16th IEEE Int. Symp. on A World of Wireless, Mobile and Multimedia Networks*. WoWMoM'15. Boston, USA: IEEE, 2015, pp. 14–17. ISBN: 978-1-4799-8461-9. DOI: 10.1109/WoWMoM.2015.7158136.

[11] U. Demir, C. Tapparello, and W. Heinzelman. "Maintaining Connectivity in Ad Hoc Networks Through WiFi Direct." In: *IEEE 14th Int. Conf. on Mobile Ad Hoc and Sensor Systems*. MASS'17. Orlando, USA, Oct. 2017, pp. 308–312. DOI: 10.1109/MASS.2017.60.

[12] U. Drolia, N. Mickulicz, R. Gandhi, and P. Narasimhan. "Krowd: A Key-Value Store for Crowded Venues." In: *10th Int. Workshop on Mobility in the Evolving Internet Architecture*. MobiArch '15. Paris, France: ACM, 2015, pp. 20–25. ISBN: 978-1-4503-3695-6. DOI: 10.1145/2795381.2795388.

[13] Y. Duan, C. Borgiattino, C. Casetti, C. F. Chiasserini, P. Giaccone, M. Ricca, F. Malabocchia, and M. Turolla. "Wi-Fi Direct Multi-group Data Dissemination for Public Safety." In: *World Telecommunications Congress*. WTC'14. Berlin, Germany: VDE, June 2014, pp. 1–6.

[14] D. Dubhashi, O. Häggström, G. Mambrini, A. Panconesi, and C. Petrioli. "Blue Pleiades, a New Solution for Device Discovery and Scatternet Formation in Multi-hop Bluetooth Networks." In: *Wireless Networks* 13.1 (Jan. 2007), pp. 107–125. ISSN: 1022-0038. DOI: 10.1007/s11276-006-1304-7. URL: http://dx.doi.org/10.1007/s11276-006-1304-7.

[15] W. Edwards. "Discovery Systems in Ubiquitous Computing." In: *Pervasive Computing, IEEE* 5 (May 2006), pp. 70 –77. DOI: 10.1109/MPRV.2006.28.

[16] Z. El-Bazzal, M. Kadoch, B. L. Agba, F. Gagnon, and M. Bennani. "A Flexible Weight Based Clustering Algorithm in Mobile Ad hoc Networks." In: *Int. Conf. on Systems and Networks Communications*. ICSNC'06. Tahiti, French Polynesia, Oct. 2006, pp. 50–50. DOI: 10.1109/ICSNC.2006.2.

[17] M. D. Felice, L. Bedogni, and L. Bononi. "The Emergency Direct Mobile App: Safety Message Dissemination over a Multi-Group Network of Smartphones Using Wi-Fi Direct." In: *14th ACM Int. Symp. on Mobility Management and Wireless Access*. MobiWac'16. Malta, Malta: ACM, 2016, pp. 99–106. ISBN: 978-1-4503-4503-3. DOI: 10.1145/2989250.2989257.

[18] C. Funai, C. Tapparello, H. Ba, B. Karaoglu, and W. B. Heinzelman. "Extending volunteer computing through mobile ad hoc networking." In: *IEEE Global Communications Conference*. GLOBECOM'14. Austin, USA, Dec. 2014, pp. 32–38. DOI: 10.1109/GLOCOM.2014.7036780.

[19] C. Funai, C. Tapparello, and W. Heinzelman. "Mobile to Mobile Computational Offloading in Multi-Hop Cooperative Networks." In: *IEEE Global Communications Conf.* GLOBECOM'16. Washington, USA, Dec. 2016, pp. 1–7. DOI: 10.1109/GLOCOM.2016.7841994.

[20] C. Funai, C. Tapparello, and W. Heinzelman. "Enabling Multi-hop Ad Hoc Networks Through WiFi Direct Multi-group Networking." In: *Int. Conf. on Computing, Networking and Communications*. ICNC'17. Silicon Valley, USA, Jan. 2017, pp. 491–497. DOI: 10.1109/ICCNC.2017.7876178.

[21] Q. Gong, Y. Guo, Y. Chen, Y. Liu, and F. Xie. "Design and Evaluation of a WiFi-Direct Based LTE Cooperative Video Streaming System." In: *IEEE Global Communications Conf.* GLOBECOM'16. Washington, USA, Dec. 2016, pp. 1–6. DOI: 10.1109/GLOCOM.2016.7841729.

[22] N. Heuveldop. *Ericsson Mobility Report*. Accessed: 2018-01-08. 2017. URL: https://www.ericsson.com/assets/local/mobility-report/documents/2017/ericsson-mobility-report-november-2017.pdf.

[23] L. V. Hoang and H. Ogawa. "A platform for building ad hoc social networks based on Wi-Fi Direct." In: *IEEE 3rd Global Conf. on Consumer Electronics*. GCCE'14. Tokyo, Japan: IEEE, Oct. 2014, pp. 626–629. DOI: 10.1109/GCCE.2014.7031171.

[24] J. Hoebeke, I. Moerman, B. Dhoedt, and P. Demeester. "An overview of mobile ad hoc networks: applications and challenges." eng. In: *Journal of the Communications Network* 3.3 (July 2004), pp. 60–66. ISSN: 1477-4739.

[25] IEEE. *802.11i-2004 - Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 6: Medium Access Control (MAC) Security Enhancements*. Accessed: 2019-07-26. June 2004. URL: https://standards.ieee.org/standard/802_11i-2004.html.

[26] IEEE. *802.11u-2011 - Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 9: Interworking with External Networks*. Accessed: 2019-07-26. Feb. 2011. URL: https://standards.ieee.org/standard/802_11u-2011.html.

[27] A. Jedda and H. T. Mouftah. "Forming MS-Free and Outdegree-Limited Bluetooth Scatternets in Pessimistic Environments." In: *IEEE Internet of Things Journal* 2.6 (2015), pp. 538–550.

[28] A. Jedda, G. Jourdan, and H. T. Mouftah. "Time-efficient algorithms for the outdegree limited bluetooth scatternet formation problem." In: *IEEE Symposium on Computers and Communications*. ISCC'12. IEEE Computer Society, July 2012, pp. 132–138. DOI: 10.1109/ISCC.2012.6249281.

[29] A. Jedda, A. Casteigts, G. Jourdan, and H. T. Mouftah. "BSF-UED: A new time-efficient Bluetooth Scatternet Formation algorithm based on Unnecessary-Edges Deletion." In: *IEEE Symp. on Computers and Communications*. ISCC'13. IEEE Computer Society, 2013, pp. 886–891.

[30] W. S. Jung, H. Ahn, and Y.-B. Ko. "Designing content-centric multi-hop networking over Wi-Fi Direct on smartphones." In: *IEEE Wireless Communications and Networking Conf.* WCNC'14. Istanbul, Turkey: IEEE, 2014, pp. 2934–2939. DOI: 10.1109/WCNC.2014.6952920.

[31] B. Kizhakedath. *Verizon makes $70 million investment for Super Bowl 50 users*. accessed: 2017-09-15. 2016. URL: http://goo.gl/QGsJKa.

[32] F. Li, S. Zhang, X. Wang, X. Xue, and H. Shen. "Vote-Based Clustering Algorithm in Mobile Ad Hoc Networks." In: *Int. Conf. on Networking Technologies for Broadband and Mobile Networks*. Vol. 3090. ICOIN'04. Springer-Verlag, 2004, pp. 13–23. ISBN: 978-3-540-25978-7.

[33] Z. Mao, J. Ma, Y. Jiang, and B. Yao. "Performance Evaluation of WiFi Direct for Data Dissemination in Mobile Social Networks." In: *22nd IEEE Symp. on Computers and Communications*. ISCC'17. Heraklion, Greece, July 2017, pp. 1213–1218.

[34] E. Marinelli. "Hyrax: Cloud Computing on Mobile Devices using MapReduce." Master's thesis. USA: School of Computer Science, CMU, 2009.

[35] R. Marinho, U. Menegato, and R. Oliveira. "IMSN Routing on Wi-Fi Direct Enabled Devices." In: *13th ACM Int. Symp. on Mobility Management and Wireless Access*. MobiWac'15. Cancun, Mexico: ACM, 2015, pp. 31–38. ISBN: 978-1-4503-3758-8. DOI: 10.1145/2810362.2810375.

[36] N. Mizumura, H. Saito, J. Takahashi, and Y. Tobe. "Towards Optimizing Time of File Transfer Among Multiple Smartphones Using Wi-Fi Direct." In: *13th Int. Conf. on Mobile and Ubiquitous Systems: Computing Networking and Services*. MobiQuitous'16. Hiroshima, Japan: ACM, 2016, pp. 189–194. ISBN: 978-1-4503-4759-4. DOI: 10.1145/3004010.3004052.

[37] A. Montresor and M. Jelasity. "PeerSim: A Scalable P2P Simulator." In: *9th IEEE Int. Conf. on Peer-to-Peer Computing*. P2P'09. Seattle, USA: IEEE, Sept. 2009, pp. 99–100. DOI: 10.1109/P2P.2009.5284506.

[38] T. Ohta, S. Inoue, and Y. Kakuda. "An Adaptive Multihop Clustering Scheme for Highly Mobile Ad Hoc Networks." In: *6th Int. Symp. on Autonomous Decentralized Systems*. ISADS'03. Pisa, Italy, Apr. 2003, pp. 293–300. DOI: 10.1109/ISADS.2003.1193960.

[39] T. Penner, A. Johnson, B. V. Slyke, M. Guirguis, and Q. Gu. "Transient clouds: Assignment and collaborative execution of tasks on mobile devices." In: *IEEE Global Communications Conf.* GLOBECOM '14. Austin, USA: IEEE, Dec. 2014, pp. 2801–2806. DOI: 10.1109/GLOCOM.2014.7037232.

[40] C. Petrioli, S. Basagni, and M. Chlamtac. "Configuring BlueStars: Multihop Scatternet Formation for Bluetooth Networks." In: *IEEE Transactions on Computers* 52.6 (June 2003), pp. 779–790. ISSN: 0018-9340. DOI: 10.1109/TC.2003.1204833.

[41] C. Petrioli, S. Basagni, and I. Chlamtac. "BlueMesh: Degree-constrained Multi-hop Scatternet Formation for Bluetooth Networks." In: *Mobile Networks and Applications* 9.1 (Feb. 2004), pp. 33–47. ISSN: 1383-469X. DOI: 10.1023/A:1027317722864.

[42] M. Pozza, C. Palazzi, and A. Bujari. "Mobile Data Offloading: An Experimental Evaluation." In: *10th Int. Workshop on Mobility in the Evolving Internet Architecture*. MobiArch'15. Paris, France: ACM, 2015, pp. 2–7. ISBN: 978-1-4503-3695-6. DOI: 10.1145/2795381.2795394.

[43] A. Pyattaev, K. Johnsson, S. Andreev, and Y. Koucheryavy. "3GPP LTE traffic offloading onto WiFi Direct." In: *IEEE Wireless Communications and Networking Conf. Workshop*. WCNCW'13. Apr. 2013, pp. 135–140. DOI: 10.1109/WCNCW.2013.6533328.

[44] J. Rodrigues, J. Silva, R. Martins, L. Lopesa, U. Drolia, P. Narasimhan, and F. Silva. "Benchmarking Wireless Protocols for Feasibility in Supporting Crowdsourced Mobile Computing." In: *16th Int. Conf. on Distributed Applications and Interoperable Systems*. DAIS'16. New York, USA: Springer-Verlag, 2016, pp. 96–108. ISBN: 978-3-319-39576-0. DOI: 10.1007/978-3-319-39577-7_8.

[45] J.-H. Ryu, S. Song, and D.-H. Cho. "New clustering schemes for energy conservation in two-tiered mobile ad-hoc networks." In: *IEEE Int. Conf. on Communications*. Vol. 3. ICC'01. 2001, 862–866 vol.3. DOI: 10.1109/ICC.2001.937361.

[46] A. A. Shahin and M. Younis. "Alert dissemination protocol using service discovery in Wi-Fi direct." In: *IEEE Int. Conf. on Communications*. ICC'15. London, UK, June 2015, pp. 7018–7023. DOI: 10.1109/ICC.2015.7249445.

[47] J. a. A. Silva, J. a. Leitão, N. Preguiça, J. a. M. Lourenço, and H. Paulino. "Towards the Opportunistic Combination of Mobile Ad-Hoc Networks with Infrastructure Access." In: *1st Workshop on Middleware for Edge Clouds and Cloudlets*. MECC'16. Trento, Italy: Association for Computing Machinery, 2016. ISBN: 9781450346689. DOI: 10.1145/3017116.3022873. URL: https://doi.org/10.1145/3017116.3022873.

[48] T. Soper. *Super Bowl fans use a record 10TB of data on Levi's Stadium wifi network, up to 63% from 2015*. Accessed: 2018-01-08. 2016. URL: http://goo.gl/hDw7Bw.

[49] A. Teófilo, D. Remédios, H. Paulino, and J. Lourenço. "Group-to-Group Bidirectional Wi-Fi Direct Communication with Two Relay Nodes." In: *12th Int. Conf. on Mobile and Ubiquitous Systems: Computing, Networking and Services*. MobiQuitous'15. Coimbra, Portugal: ACM, July 2015, pp. 275–276. ISBN: 978-1-63190-072-3. DOI: 10.4108/eai.22-7-2015.2260272.

[50] A. Teófilo, D. Remédios, H. Paulino, and J. Lourenço. "Comunicação Móvel Inter-Grupo Baseada em TCP sobre Wi-Fi Direct." In: *8th INForum - Simp. de Informática*. INFORUM'16. Lisboa, Portugal, 2016.

[51] A. Teófilo, D. Remédios, J. M. Lourenço, and H. Paulino. "GOCRGO and GOGO: Two Minimal Communication Topologies for WiFi-Direct Multi-group Networking." In: *14th Int. Conf. on Mobile and Ubiquitous Systems: Computing, Networking and Services*. MobiQuitous'17. Melbourne, Australia: ACM, 2017, pp. 232–241. ISBN: 978-1-4503-5368-7. DOI: 10.1145/3144457.3144481.

[52] R. Trestian, Q.-T. Vien, H. X. Nguyen, and O. Gemikonakli. "ECO-M: Energy-efficient Cluster-Oriented Multimedia delivery in a LTE D2D environment." In: *IEEE Int. Conf. on Communications*. ICC'15. London, UK, June 2015, pp. 55–61. DOI: 10.1109/ICC.2015.7248298.

[53] S. Trifunovic, B. Distl, D. Schatzmann, and F. Legendre. "WiFi-Opp: Ad-hoc-less Opportunistic Networking." In: *6th ACM Workshop on Challenged Networks*. CHANTS'11. Las Vegas, Nevada, USA: ACM, 2011, pp. 37–42. ISBN: 978-1-4503-0870-0. DOI: 10.1145/2030652.2030664.

[54] Z. Wang, R. J. Thomas, and Z. J. Haas. "Bluenet - A New Scatternet Formation Scheme." In: *35th Annual Hawaii Int. Conf. on System Sciences*. HICSS'02. Big Island, Hawaii: IEEE Computer Society, 2002. DOI: 10.1109/HICSS.2002.993971.

[55] Wi-Fi Alliance. *Wi-Fi Protected Setup Specification*. Version 1.0h. Dec. 2006.

[56] Wi-Fi Direct. *Wi-Fi Direct: Portable Wi-Fi that goes with you anywhere @ONLINE*. Accessed: 2018-01-08. June 2014. URL: http://www.wi-fi.org/discover-wi-fi/wi-fi-direct.

[57] P. Wong, V. Varikota, D. Nguyen, and A. Abukmail. "Automatic Android-based Wireless Mesh Networks." In: *Informatica (Slovenia)* 38.4 (2014), pp. 313–320.

[58] X. Xu, Z. M. Mao, and J. A. Halderman. "Internet Censorship in China: Where Does the Filtering Occur?" In: *Passive and Active Measurement*. Ed. by N. Spring and G. F. Riley. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 133–142. ISBN: 978-3-642-19260-9.

[59] N. Zaguia, Y. Daadaa, and I. Stojmenovic. "Simplified Bluetooth Scatternet Formation Using Maximal Independent Sets." In: *Integrated Computer-Aided Engineering* 15.3 (Aug. 2008), pp. 229–239. ISSN: 1069-2509. DOI: 10.3233/ICA-2008-15303.

[60] G. Zaruba, S. Basagni, and I. Chlamtac. "Bluetrees-scatternet formation to enable Bluetooth-based ad hoc networks." In: *IEEE Int. Conf. on Communications*. ICC'01. Helsinki, Finland: IEEE, 2001, pp. 273–277. DOI: 10.1109/ICC.2001.936316.