Cairo University
Faculty of Engineering
Department of Computer Engineering

C4IAN

A Graduation Project Report Submitted

to

Faculty of Engineering, Cairo University

in Partial Fulfillment of the requirements of the degree

of

Bachelor of Science in Computer Engineering

Presented by

Abdulrahman Khalid Hassan
Ahmed Mahmoud AbdEl-Monaem
Mahmoud Othman Adas
Yosry Mohammad Yosry

Supervised by

Dr. Ahmed Hamdy

July 24, 2021

Abstract

Mobile Ad-hoc Networks (MANETs) is an ever-growing research field due to their numerous practical applications. MANETs provide means for infrastructure-less communications, they are utilized by tactical teams, military, emergency and rescue teams, drone swarms, and many others.

C4IAN (Command, Control, Communications, Computing & Intelligence using Ad-hoc Networks) is a scalable communications solution for tactical teams. It builds a MANET on-the-go between command centers and units, allowing them to communicate effectively and securely while moving freely, without the need for any infrastructure.

Our system can be run on handheld tactical devices, robots, drones, or any other Linux-based platform. It features both a platform-independent network layer which routes data through dynamically changing and scalable topologies, and applications for tactical teams' units and command centers to communicate through.

C4IAN's router implements Zone Hierarchical Link state Routing (ZHLS) routing for uni-casting, On Demand Multicast Routing Protocol (ODMRP) for multi-casting, and a unique Variable-Radius Broadcasting (VRB) that is based on ZHLS. We also introduced some improvements to ZHLS and ODMRP to optimize their implementation.

المُلَخَّص

الشبكات التلقائية المتحركة تلقي رواجاً كبيرة كمجال للبحث العلمي حيث أن لها عدداً كبيراً من التطبيقات العملية. الشبكات التلقائية المتحركة توفر طريقة للتواصل بدون بنية تحتية، و لها مستخدمون عديدون، مثل الفرق التكتيكية، و الجيوش، وفرق الطوارئ و الإنقاذ، و الروبوتات، و غيرهم.

كيان هو نظام تواصل معتمد للفرق التكتيكية. كيان يقوم ببناء شبكة تلقائية عند الحاجة بين مراكز التحكم و الوحدات، ممكناً إياهم من التواصل بشكل سريع و آمن أثناء التحرك بحرية، بدون الحاجة إلي بنية تحتية.

نظام كيان يمكن تشغليه علي أجهزة التواصل المحمولة الخاصة بالفرق التكتيكية، أو علي روبوتات، أو علي أي جهاز يدعم نظام تشغيل لينكس. النظام قادر علي تحديد مسار البيانات و توجيهها داخل الشبكات كبيرة الحجم المتغيرة بسرعة، بالإضافة إلي برنامجين لوحدات الفرق التكتيكية و مراكز القيادة.

موجه البيانات في كيان يستخدم بروتوكول توجيه هرمي مبني على المناطق للبيانات الموجهة لوجهة واحدة، و بروتوكل توجيه متعدد عند الحاجة للبيانات الموجهة لعدة جهات، و بروتوكول نشر المعلومات لقطر معين لتوجيه البيانات الموجهة للجميع. لقد قمنا أيضاً بإضافة بعض التحسينات علي هذه البروتوكولات.

Acknowledgment

Our gratitude to those who helped us cannot be put in words. First, we would like to express our appreciation and thankfulness to our supervisor, **Dr. Ahmed Hamdy**, for his utmost care and support for us during our work. His guidance and mentorship have been critical to our success.

We would also like to thank our family, friends and colleagues, who have always been there to support us and push us forward. We are defined by the people who surround us, and those people made us who we are today.

Contents

Abstract . . . ii
Acknowledgment . . . iv
Contents . . . viii
List of Figures . . . x
List of Tables . . . xi
List of Abbreviations . . . xii
Contacts . . . xiv

1 Introduction **1**
1.1 Motivation and Justification . . . 1
1.2 Project Objectives and Problem Definition . . . 2
1.3 Project Outcomes . . . 2
1.4 Document Organization . . . 2

2 Market Feasibility Study **3**
2.1 Targeted Customers . . . 3
2.2 Market Survey . . . 3
2.2.1 Bittium . . . 3
2.2.2 Hytera . . . 4
2.3 Business Case and Financial Analysis . . . 4
2.3.1 Business Case . . . 5
2.3.2 Financial Analysis . . . 5

3 Literature Survey **8**
3.1 Unicast Routing in MANETs . . . 8
3.2 Comparative study of Unicasting . . . 9
3.2.1 Proactive routing protocols . . . 9
3.2.1.1 Destination Sequenced Distance Vector (DSDV) . . . 9
3.2.1.2 Optimized Link State Routing Protocol (OLSR) . . . 9
3.2.1.3 Fisheye State Routing (FSR) . . . 10
3.2.2 Reactive routing protocols . . . 10
3.2.2.1 Ad Hoc On-Demand Distance Vector Routing Protocol (AODV) 10
3.2.2.2 Dynamic Source Routing (DSR) . . . 11
3.2.3 Hybrid Routing Protocols . . . 12
3.2.3.1 Zone Routing Protocol (ZRP) . . . 12
3.2.3.2 Zone Hierarchical Link state Routing (ZHLS) . . . 13
3.2.4 Overall Comparison of All Unicast Routing Protocols . . . 13
3.3 Implemented Unicast Routing: ZHLS . . . 13
3.4 Multicast Routing in MANETs . . . 15

3.4.1 Classification Of Multicast Routing Protocol . . . 16
3.4.1.1 Multicast Topology . . . 16
3.4.1.2 Routing Initialization Approach . . . 17
3.4.1.3 Routing Scheme . . . 17
3.4.1.4 Multicast Maintenance Approaches . . . 18
3.5 Comparative study of Multicasting . . . 18
3.5.1 Multicast routing protocols . . . 19
3.5.2 Performance Evaluation Of Multicast Routing Protocols . . . 19
3.5.2.1 Scenario is varying the number of nodes . . . 19
3.5.2.2 Varying the speed of nodes . . . 20
3.6 Implemented Multicast Routing: ODMRP . . . 23
3.6.1 ODMRP Overview . . . 23

4 System Design and Architecture 25
4.1 Overview and Assumptions . . . 25
4.1.1 Reliability . . . 25
4.1.2 Speed . . . 26
4.1.3 Security . . . 26
4.2 System Architecture . . . 26
4.2.1 Nodes . . . 26
4.2.1.1 Units . . . 26
4.2.1.2 Command Centers . . . 27
4.2.2 Block Diagram . . . 27
4.3 The Router . . . 28
4.3.1 Functional Description . . . 29
4.3.2 Modular Decomposition . . . 29
4.3.2.1 Kernel Interface . . . 30
4.3.2.2 Forwarder . . . 30
4.3.2.3 Variable Radius Broadcast (VRB) . . . 31
4.3.2.4 Flooder . . . 32
4.3.2.5 M-Sec . . . 33
4.4 Unicast Controller . . . 33
4.4.1 Functional Description . . . 33
4.4.2 Modular Decomposition . . . 34
4.4.2.1 Zone Agent . . . 35
4.4.2.2 Secure ARP . . . 36
4.4.2.3 Hierarchical LSR . . . 38
4.4.2.4 Hierarchical Topology Graph . . . 39
4.4.2.5 Destination Zone Discovery . . . 40
4.5 Multicast Controller . . . 42
4.5.1 Functional Description . . . 42
4.5.2 Modular Decomposition . . . 42
4.5.2.1 Control Packets Generator . . . 42
4.5.2.2 Tables . . . 45
4.5.3 ODMRP Protocol Flow . . . 48
4.6 Command Center Client . . . 48
4.6.1 Functional Description . . . 49
4.6.2 Modular Decomposition . . . 49

4.6.2.1 Daemon . . . 50
4.6.2.2 Video Streamer . . . 50
4.6.2.3 User Interface . . . 51
4.7 Unit Client . . . 52
4.7.1 Functional Description . . . 52
4.7.2 Modular Decomposition . . . 54
4.7.3 Design Constraints . . . 54

5 System Testing and Verification 55
5.1 Testing Setup . . . 55
5.1.1 Mininet-Wifi . . . 55
5.1.1.1 Installing Mininet-Wifi . . . 56
5.1.2 Edit GUI and MN Script . . . 57
5.1.3 Start Script . . . 57
5.1.3.1 Socat . . . 58
5.2 Testing Plan and Strategy . . . 58
5.2.1 Module Testing . . . 59
5.2.2 Integration Testing . . . 59
5.3 Comparative Results to Previous Work . . . 61

6 Conclusions and Future Work 62
6.1 Faced Challenges . . . 62
6.1.1 MANET routing . . . 62
6.1.2 Interfacing with Linux Kernel . . . 62
6.1.3 Testbed . . . 63
6.1.4 Testing and Debugging . . . 63
6.1.5 Performance Metrics . . . 63
6.2 Gained Experience . . . 63
6.3 Conclusions . . . 64
6.4 Future Work . . . 64
6.4.1 Hardware Deployment . . . 64
6.4.2 Multicast Support for TCP . . . 64
6.4.3 Alternatives to Mininet-WiFi . . . 64
6.4.4 Uniform Shapes for Hierarchical Zones . . . 65
6.4.5 Implementing The Router as a Kernel Module . . . 65
6.4.6 Forwarding with eBPF . . . 65
6.4.7 No FFmpeg . . . 66
6.4.8 Different Quality Levels of Streaming . . . 66

Bibliography 67

Appendices 70

A Development Platforms and Tools 71
A.1 Hardware Tools . . . 71
A.2 Software Tools . . . 71
A.2.1 Programming Languages . . . 71
A.2.1.1 Golang . . . 71
A.2.1.2 Python . . . 71

A.2.1.3 Shell Scripting . . . 71
A.2.2 Libraries and Frameworks . . . 72
A.2.2.1 Mininet . . . 72
A.2.2.2 Socat . . . 72
A.2.2.3 Nsenter . . . 72
A.2.2.4 React.js . . . 72
A.2.2.5 Electron.js . . . 72
A.2.3 Tools and Platforms . . . 72

B Use Cases **73**

C User Guide **74**
C.1 Environment Setup . . . 74
C.1.1 Golang 1.15.8 . . . 74
C.1.2 python packages . . . 75
C.1.3 Libnetfilter-queue . . . 75
C.1.4 Known Issues . . . 75
C.1.4.1 Electron with libgconf . . . 75
C.1.4.2 Writing to `/tmp/` . . . 75
C.2 System Deployment . . . 75
C.2.0.1 Router Deployment . . . 76
C.2.0.2 Unit Deployment . . . 76
C.2.0.3 Command-Center Daemon Deployment . . . 76
C.2.0.4 Command-Center UI Deployment . . . 77
C.3 Usage . . . 77
C.3.1 Router . . . 77
C.3.2 Unit . . . 77
C.3.3 Command Center UI . . . 77

D Feasibility Study **82**
D.1 Technical Feasibility . . . 82
D.2 Operational Feasibility . . . 82
D.3 Legal Feasibility . . . 83
D.4 SWOT Analysis . . . 83

List of Figures

3.1 Multipoint relays . . . 10
3.2 Fisheye hierarchical scopes . . . 11
3.3 AODV route discovery . . . 11
3.4 DSR packets . . . 12
3.5 ZHLS network structure . . . 13
3.6 classification of multicast routing protocols . . . 16
3.7 Changing number of nodes vs reliability, throughput, delay and packet drop ratio . . . 21
3.8 Changing speed of nodes vs reliability, throughput, delay and packet drop ratio 22

4.1 Overall System . . . 28
4.2 The Router Modules . . . 29
4.3 Data Flow Through the Forwarder . . . 30
4.4 Flooding Header . . . 32
4.5 ZHLS Modules . . . 34
4.6 Zones as Perceived by Node of Different ZLengths . . . 35
4.7 ZID Header . . . 35
4.8 SARP Header . . . 36
4.9 SARP Protocol . . . 37
4.10 Intrazone LSR . . . 38
4.11 Interzone LSR . . . 39
4.12 Hierarchical Topology Graph . . . 40
4.13 DZD Request Header . . . 41
4.14 DZD Response Header . . . 42
4.15 ODMRP Modules . . . 43
4.16 Join Query Packet . . . 44
4.17 Join Reply Packet . . . 45
4.18 Group Members Table . . . 45
4.19 Cache Table Entry . . . 46
4.20 Members Table Entry . . . 46
4.21 Forwarding Table Entry . . . 47
4.22 Multi Forward Table Entry . . . 47
4.23 Command Center Client . . . 49
4.24 Command Center Client Daemon . . . 50
4.25 Video Streamer Data Flow . . . 51
4.26 CMD UI Login . . . 52
4.27 CMD UI Map . . . 52
4.28 CMD UI Units . . . 53

4.29 CMD UI Streams 53
4.30 Unit Modules 54

5.1 Mininet Testbed With Different Nodes 56
5.2 Edit GUI 57
5.3 Socat Forwarding and UI 58
5.4 Testing Strategy 59
5.5 Module Testing 60
5.6 Log Visualizer 60

C.1 CMD UI Filling Port Number 77
C.2 CMD UI Map Page 78
C.3 CMD UI Menu 78
C.4 CMD UI Broadcast Button 79
C.5 CMD UI Broadcast Dialog 79
C.6 CMD UI Units Page 79
C.7 CMD UI Code Message (Attack) 80
C.8 CMD UI Requesting Video Streaming From Unit 80
C.9 CMD UI Streaming Page 81

D.1 SWOT Analysis 83

List of Tables

2.1 Business case over 5 years 5
2.2 Capex Table 5
2.3 Opex Table 6
2.4 Cache Flow Table in (EGP) 6

3.1 Summary Comparison of Unicast Routing Protocols 14

5.1 Summary of Differences Between Our Work and Previous Work 61

List of Abbreviations

AODV Ad Hoc On-Demand Distance Vector Routing Protocol. v, ix, 10, 11, 14, 19

API Application Programming Interface. 50, 51

ARP Address Resolution Protocol. vi, 36, 38, 39

C4IAN Command, Control, Communications, Computing & Intelligence using Ad-hoc Networks. ii, 1, 2, 27, 32, 33, 36, 40, 49, 55, 58, 59, 62–65

CMD Command Center Client. ix, x, 25, 26, 52, 53

DMR Digital Mobile Radio. 4

DSDV Destination Sequenced Distance Vector. v, 9, 11

DSR Dynamic Source Routing. v, ix, 11, 12, 14

eBPF Extended Berkeley Packet Filter. vii, 65, 66

FSR Fisheye State Routing. v, 10

GUI Graphical User Interface. 25, 63

HAL Hardware Abstraction Layer. 54, 57, 60

HLS Http Live Streaming. 66

HW Hardware. 25, 54

IP Internet Protocol. 28–34, 36, 37, 39–42, 44–48, 62

MAC Media Access Control. 28, 30, 31, 36–38, 44, 46, 48

MANET Mobile Ad-hoc Network. ii, vii, 1, 2, 4, 8, 16, 62, 63

MANETs Mobile Ad-hoc Networks. ii, v, 1–3, 8, 9, 15, 62–64

ODMRP On Demand Multicast Routing Protocol. ii, vi, 8, 18–20, 23, 24

OLSR Optimized Link State Routing Protocol. v, 9

PoC Push-to-Tablk over Cellular. 4

R&D Research and Development. 3

REST Representational State Transfer. 50, 51

RFC Reference for Comments. 63

SARP Secure Address Resolution Protocol. ix, 36–38

SSE Server-Side Events. 50, 51

UI User Interace. ix, x, 26, 27, 52, 53

utils utilities. 59

VoIP Voice Over IP. 4

VRB Variable-Radius Broadcasting. ii

ZHLS Zone Hierarchical Link state Routing. ii, v, ix, 8, 12–15, 29, 32, 34–36, 38–40, 59

ZID Zone IDentification Protocol. ix, 29, 31, 32, 35–37, 41, 65

ZRP Zone Routing Protocol. v, 12

Contacts

Team Members

Name	Email	Phone Number
Abdulrahman Khalid Hassan	abdulrahman.elshafei98@gmail.com	+2 01011315175
Ahmed Mahmoud AbdEl-Monaem	ahmed.afifi.cufe@gmail.com	+2 01003242713
Mahmoud Othman Adas	mahmoud.othman.adas@gmail.com	+2 01019426003
Yosry Mohammad Yosry	yosrym93@gmail.com	+2 01149392716

Supervisor

Name	Email	Phone Number
Dr. Ahmed Hamdy	ahamdy@eng.cu.edu.eg	+2 01012697605

This page is intentionally left blank

Chapter 1

Introduction

An ad-hoc network is a decentralized network that does not rely on pre-existing infrastructure. There are no dedicated routers or access points. Instead, hosts (user devices) act as both a user and a router in the network. When a message is sent, it gets forwarded between different nodes in the network until it reaches its destination.[33] [25]

Mobile Ad-hoc Networks (MANETs) are a type of wireless ad-hoc networks where the devices can move freely. This increases the versatility of the network, and extends its use cases to more practical applications. Nonetheless, the routing problem becomes much more complex as the topology of the network is changing dynamically. Links between devices are frequently broken, and devices need to be up-to-date with the most recent version of the topology to be able to communicate. Furthermore, as the network size increases, keeping track of such a dynamic topology becomes even harder.[19]

Naturally, MANETs have always been an active area of research due to the complexity of the routing problem. Different approaches to the routing problem are being researched continuously, with very few practical implementations.

C4IAN (Command, Control, Communications, Computing & Intelligence using Ad-hoc Networks) is a scalable communications solution for tactical teams. It builds a MANET on-the-go between command centers and units, allowing them to communicate effectively and securely while moving freely, without the need for any infrastructure.

1.1 Motivation and Justification

Tactical teams, such as emergency and rescue teams, police officers, and military battalions, require a robust communications solution. They cannot rely on infrastructure-based communications (such as cellular networks) for multiple reasons:

- They can be damaged during an emergency or a catastrophe.
- They are unavailable in remote areas.
- They can be easily targeted by enemies.
- Commercial ones may not be secure enough.

Hence, MANETs are usually the obvious means of communications for such tactical teams. C4IAN aims to provide a software MANET-based communications solution for tactical teams,

that can be deployed on any Linux platform.

1.2 Project Objectives and Problem Definition

We aim to build a scalable software system that provides reliable communications between devices that are moving freely, without any infrastructure. The core of such a system is routing. Our core objective is to develop a scalable MANET routing solution that will be the basis of a concrete communications system.

MANET routing is an open-ended research problem. There are no established protocols that work for all use cases. C4IAN implements unicast, multicast, and broadcast routing. For each type of routing, there are many protocols in the research literature, with very few implementations. We surveyed many protocols, selected and tuned adequate ones, and morphed the results of theoretical research into concrete practical implementations.

1.3 Project Outcomes

The outcome of the project is two-fold. The core part is a Linux-based network layer that implements unicast, multicast, and broadcast routing. The customer-facing part is an application that utilizes this network layer to provide communications to command centers and units in a tactical team.

1.4 Document Organization

In this chapter, we provided an introduction to MANETs and to our communications system for tactical teams, C4IAN. The rest of the document is organized as follows:

- In chapter 2 we discuss the potential market for our product, and the competition we may face.
- In chapter 3 we review different approaches to unicast and multicast routing in MANETs, and lay the necessary background to understand the routing problem.
- In chapter 4 we explain how our system works. We dive deep into the architecture of C4IAN and discover what every module contributes.
- In chapter 5 we explain how we tested different aspects of our system, and how it compares to related work.
- In chapter 6 we conclude the document by describing the challenges we faced, the lessons that we learned, and what the future may look like for C4IAN.

Afterwards, the appendices outline the development platforms and tools that we used, different use cases of C4IAN, the user guide, code documentation, and a comprehensive feasibility study.

Chapter 2

Market Feasibility Study

Achieving scalable and reliable communications in areas that lack network infrastructure is a difficult problem, especially in critical situations such as the battlefield or during emergencies. It is also needed for collecting real time data and analyze it to have a competitive advantage in the battleground.
Mobile Ad-hoc Networks (MANETs) can be formed on-the-go, without any infrastructure. They promise more flexibility and reliability than manual radio broadcasting.

2.1 Targeted Customers

Our initial market will be the tactical teams by providing tactical system and its products for defense industry, rescue teams and police we will provide in yearly subscription for using our products, training and support. As our team gets sophisticated enough, our target customer base will widen, and we will build another solution to fit a normal customer using the same router module. We will build a communication/streaming/gaming service and target customers who do not have internet infrastructure in their area, by providing additional servers in their areas which will act as routers and have significant caches of movies and web data.

2.2 Market Survey

2.2.1 Bittium

Bittium [5] specializes in the development of reliable, secure communications and connectivity solutions with experience over 35 years in advanced radio communication technologies. Bittium provides innovative products and services, customized solutions based on its product platforms and Research and Development (R&D) services. Complementing its communications and connectivity solutions, Bittium offers proven information security solutions for mobile devices and portable computers. Bittium also provides healthcare technology products and services for biosignal measuring in the areas of cardiology, neurology, rehabilitation, occupational health and sports medicine. Net sales in 2020 were EUR 78.4 million and operating profit was EUR 2.1 million.

Bittium offers communication options, which are critical in combat. Bittium TAC WIN, the company's primary competing product, is a high-performance wireless network solution

that enables fast and flexible link, point-to-multipoint, and MANET (Mobile Ad Hoc Network) connections.
The system enables improved connection to current wired and wireless IP infrastructures and legacy systems, as well as broadband IP data transmission for mobility troops in all areas of the battlefield.

The Estonian Defense Forces requested a tactical communication network based on the Bittium TAC WIN system.
The TAC WIN system will be used to modernize IP data transmission, improve the performance of the tactical data transmission, and diversify the wireless and cable connections of the Estonian Land Forces' system entity.
The Bittium Tough Voice Over IP (VoIP) product family's products will be used to upgrade the Land Forces' IP voice service, and when combined with the Bittium TAC WIN system, it forms a software solution that seamlessly integrates the tactical network and its voice services for leading troops in the most demanding situations.

On July 1, 2021, Bittium has received another purchased order from the Finnish Defense Forces for the development of a new software version for the software defined radio based Bittium Tactical Wireless IP Network system.
The purchase order is worth roughly 2.3 million EUR (excl. VAT), with extra purchases with a maximum value of 0.5 million EUR (excl. VAT) are included in the procurement decision. The development work will be completed by the end of the first half of 2022.

2.2.2 Hytera

Hytera [20] is a Chinese manufacturer of radio transceivers and radio systems. Hytera's sales and revenue in 2020 is 6.06 Billion dollars [12].

Hytera provides two types of products Digital Mobile Radio (DMR) and Push-to-Tablk over Cellular (PoC) systems and applications. The main competitor product is Hytera HALO Dispatch which is a Push-to-Tablk over Cellular (PoC) radio systems that enables countrywide or single-site communications and dispatching, as well as rapid group calling that supports data, audio, and video. The three systems are capable of functioning as stand-alone networks.
Hytera Halo Dispatch Software is billed annually. The first year is $999, each additional year is$599.00.

2.3 Business Case and Financial Analysis

The target product of our company is to produce the entire system from A to Z by request. The unit cost is estimated to range from 3000 to 5000 EGP, including its testing. The command center device cost is estimated to range from 8,000 to 10,000 EGP including its testing. In this section, two aspects will be addressed:

1. Business Case.

2. Financial Analysis.

2.3.1 Business Case

In the business case, we will illustrate the projected amount of product sales over the next five years, which is shown in Table 2.1, as well as how we will adjust the pricing to compete.

Years	Year 1	Year 2	Year 3	Year 4	Year 5
CMD Ordered	1,000	2,000	3,000	5,000	10,000
Units Ordered	10,000	15,000	20,000	50,000	80,000
CMD Cost	10,000	10,100	10,300	10,500	10,550
Unit Cost	4,000	4,200	4,400	4,600	4,650
Purchased CMD Cost	22,000 EGP	25,000 EGP	30,000 EGP	40,000 EGP	41,000 EGP
Purchased Unit Cost	10,000 EGP	12,000 EGP	15,000 EGP	17,000 EGP	17,500 EGP
Profit From Purchasing	72M EGP	146.8M EGP	271.1M EGP	767.5M EGP	1.3325B EGP
Profit From Maintenance	2M EGP	4M EGP	6M EGP	10M EGP	20M EGP
Total Profit	74M EGP	150.8M EGP	277.1M EGP	777.5M EGP	1.3525B EGP

Table 2.1: Business case over 5 years

2.3.2 Financial Analysis

Financial analysis is done based on the company's business case to predict the Capital Expenses (Capex) and the Operational Expenses (Opex).
Table 2.2 displays the Capex and Table 2.3 shows the Opex.

Items And Supplies	Cost
Offices, chairs, air conditioners, security cameras, routers and switches	90,000 EGP
Developers Laptops	10 * 15 = 150,000 EGP
30 units of Raspberry Pi 4 Model B 4 GB for testing	1,720.00 * 30 = 51,600 EGP
30 units of RC Waterproof 2.4Ghz All Terrain Off-Road Amphibious Car	600 * 30 = 18,000 EGP
Total Capex	409,600 EGP

Table 2.2: Capex Table

As can be seen in the Cash Flow Table 2.4, it is expected that we do not sell any units or command center devices for the first four months. These four months are dedicated for testing and training troops. We should break even by the beginning of May with a cumulative

Items And Supplies	Cost per Month
Employees' salaries	150,000 EGP
Office Rent	4,000 EGP
Training customers and testing fields Rent	12,000 EGP
Office maintenance	2,000 EGP
Servers maintenance	3,000 EGP
Bills (electricity, water, gas, etc.)	2,000 EGP
Total Opex	173,000 EGP

Table 2.3: Opex Table

Money Going Out												
Month	1	2	3	4	5	6	7	8	9	10	11	12
Capex												
Office	7,500	7,500	7,500	7,500	7,500	7,500	7,500	7,500	7,500	7,500	7,500	7,500
Laptops	12,500	12,500	12,500	12,500	12,500	12,500	12,500	12,500	12,500	12,500	12,500	12,500
30 Unit Raspberry Pi	4,300	4,300	4,300	4,300	4,300	4,300	4,300	4,300	4,300	4,300	4,300	4,300
30 Unit Car	1,500	1,500	1,500	1,500	1,500	1,500	1,500	1,500	1,500	1,500	1,500	1,500
Opex												
Salaries	150,000	150,000	150,000	150,000	150,000	150,000	150,000	150,000	150,000	150,000	150,000	150,000
Office Rent	4,000	4,000	4,000	4,000	4,000	4,000	4,000	4,000	4,000	4,000	4,000	4,000
Fields Rent	12,000	12,000	12,000	12,000	12,000	12,000	12,000	12,000	12,000	12,000	12,000	12,000
Office Maintenance	2,000	2,000	2,000	2,000	2,000	2,000	2,000	2,000	2,000	2,000	2,000	2,000
Servers Maintenance	3,000	3,000	3,000	3,000	3,000	3,000	3,000	3,000	3,000	3,000	3,000	3,000
Bills	2,000	2,000	2,000	2,000	2,000	2,000	2,000	2,000	2,000	2,000	2,000	2,000
Total Out	198,800	198,800	198,800	198,800	198,800	198,800	198,800	198,800	198,800	198,800	198,800	198,800
Money Going In												
Total CMD & Unit Purchased Profits	0	0	0	0	925,000	925,000	925,000	925,000	925,000	925,000	925,000	925,000
Maintainance	0	0	0	0	500,00	500,00	500,00	500,00	500,00	500,00	500,00	500,00
Total In	0	0	0	0	1.425 Million	1.425 Million	1.425 Million	1.425 Million	1.425 Million	1.425 Million	1.425 Million	1.425 Million
Profit	-198,800	-198,800	-198,800	-198,800	1.2262 Million	1.2262 Million	1.2262 Million	1,226,200	1,226,200	1,226,200	1,226,200	1,226,200
Taxes (25%)	0	0	0	0	306,550	306,550	306,550	306,550	306,550	306,550	306,550	306,550
Net Profit	-198,800	-198,800	-198,800	-198,800	919,650	919,650	919,650	919,650	919,650	919,650	919,650	919,650
Cumulative Profit	-198,800	-397,600	-596,400	-795,200	124,450	1,044,100	1,963,750	2,883,400	3,803,050	4,722,700	5,642,350	6,562,000

Table 2.4: Cache Flow Table in (EGP)

profit value of **124,450 EGP**. The main profit contribution comes from the purchase, not the support.
This means that depending only on maintenance is not a good strategy, and that we should always keep our customers happy with reasonable prices and excellent services.
Our product is the only one of its kind in the Middle East. Moreover, our product is so much cheaper than our competitors outside the Middle East. Hytera Halo Dispatch Software mentioned in the previous section is an example. It costs $999 per unit per year. They also adopt a different business model and they depend more on the annual subscription.
In our case, we build the system unit or CMD and sell it one time with a maintenance subscription, which is so much cheaper than selling annual subscription used by other companies.

Chapter 3

Literature Survey

This section will present an overview of the literature on routing protocols in MANET. An ad-hoc routing protocol is a convention or standard that regulates how nodes choose which method to route data between computing devices in a mobile ad-hoc network. According to the desired destination, ad-hoc network routing techniques are split into three types: unicast routing for a single destination, multicast routing for a group of destinations, and lastly broadcast routing for all targets in the network.

A comprehensive discussion of unicast routing protocols in MANET, particularly ZHLS (our implemented approach), is given in sections 3.1, 3.2, and 3.3. In sections 3.3, 3.4, and 3.5, we will discuss multicast routing protocols, particularly ODMRP (our implemented approach).

3.1 Unicast Routing in MANETs

The research interest in mobile ad-hoc networking grew rapidly in the 1990s. Because of the network's lack of infrastructure and dynamic nature, new networking methods must be adopted in order to ensure efficient end-to-end communication. This, combined with the numerous applications of these networks in various circumstances such as the battlefield and disaster recovery, has resulted in MANETs routing being studied by a variety of organizations and institutes. The majority of MANET applications rely on unicast connectivity. The source mobile node transmits data packets to the destination in unicast mode. The destination address in the data packet is used by the dispatch node to search it up in the routing table when forwarding data packets[32].

It's not easy to forward packets in a MANET with restricted resources (such as bandwidth restrictions, concealed terminals, and low battery power). To do this, a number of protocols have been developed. To decrease packet loss, communication overheads, and enhance data delivery ratio, various MANET unicast routing protocols have concentrated on attaining stability and dependability. Other unicast routing protocols have placed a premium on scalability to be capable of dealing with huge networks with minimum computations. Other protocols placed an emphasis on data transmission speed and the ability to meet real-time constraints. To attain those objectives, many techniques have been developed. A comparative study of these Unicast routing techniques will be proposed in the next section.

3.2 Comparative study of Unicasting

The fundamental goal of ad-hoc routing protocols is to efficiently transmit data packets between nodes without relying on preset topologies or centralized management. Any unicast routing protocol must have the following characteristics:

- It should be assumed that routes are unidirectional.
- It should consider data security.
- It must be energy-and-power-efficient.
- It should be distributed in such a way as to enhance its dependability.

Based on the previous properties, unicast routing techniques in MANETs can be divided into three classes based on the time it takes for nodes to obtain a route to a destination. Those classes are reactive, proactive, and hybrid.

3.2.1 Proactive routing protocols

Proactive routing protocols, also known as table-driven routing protocols, require each node to maintain one or more tables containing up-to-date and consistent routing information for all other nodes in the network. these protocols learn the network's global topology by sharing information across network nodes. The nodes send out update messages every time the topology changes, and the knowledge about this change is disseminated across the network. Using this information, the shortest path can be computed from each source to any destination, but on the other side a lot of bandwidth is consumed[30]. In the following sections, some proactive routing protocols will be discussed.

3.2.1.1 Destination Sequenced Distance Vector (DSDV)

The Destination Sequenced Distance Vector (DSDV) is an extension of the well-known wired routing Distance Vector protocol. DSDV addresses the primary problem associated with distance vector routing protocol in wired networks by utilizing destination sequence numbers, namely count-to-infinity. old routes and new routes are distinguished using sequence numbers originated at the destination, enabling the routing protocol to avoid route loops. multiple routes with the same sequence number are distinguished using the shortest route metric.[1]

3.2.1.2 Optimized Link State Routing Protocol (OLSR)

Optimized Link State Routing Protocol (OLSR) is an also extension and optimized variant of the well-known pure link-state routing protocol. It uses a multipoint relaying technique to efficiently transmit link-state packets. As the traditional link-state routing protocol, by sending link-state messages on a regular basis, each node keeps track of the network's topology. The optimization is mostly accomplished in two ways. First, during each route update, the size of the control packets for a specific node is reduced heavily by exchanging just a subset of links with the node's neighbors who are its multipoint relay selectors, rather than all links in the network. Each packet may be read and processed by any node not in the set, but it cannot be retransmitted. Second, it reduces control traffic flooding by employing just chosen nodes, known as multipoint relays, to distribute information in the network. Each node broadcasts a list of its one-hop neighbors to select these multipoint relaying nodes. A subset of these one-hop neighbors are chosen from the list of nodes in the hello messages that covers all of

its two-hop neighbors. Node A in Figure 3.1 can choose MPR nodes from nodes B, C, K, and N. Because these nodes cover all nodes that are two hops apart.[9] [8]

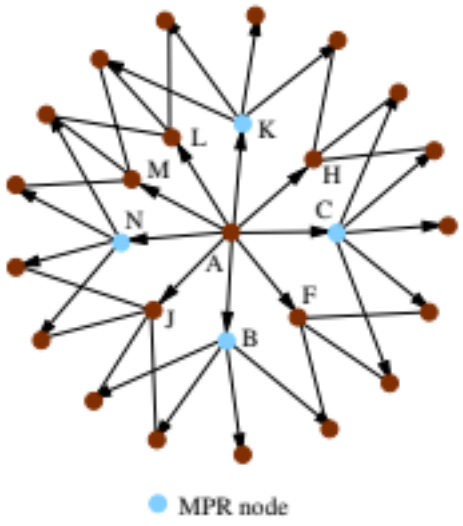

Figure 3.1: Multipoint relays

3.2.1.3 Fisheye State Routing (FSR)

Fisheye State Routing (FSR) is another optimized variant of the link-state routing protocol. It effectively reduces the amount of information and time it takes to keep network topology up to date. Link state updates are created and flooded across the network anytime a topology change is detected in the traditional link state routing method. However, in FSR, nodes only communicate link status information on a periodic basis. By updating network information for local nodes at a higher frequency than for faraway nodes beyond the fisheye scope, FSR lowers the size of update messages. As a result, FSR is more scalable for huge networks. Figure 3.2 illustrate the FSR hierarchical scopes.[28]

3.2.2 Reactive routing protocols

On demand routing systems are also known as reactive protocols. By preserving route information for active routes, such protocol decreased the overheads of proactive protocol. It means that routing information is only necessary and stored when a node wishes to transmit a data packet to a specific destination. From the previous description, we can deduce that the main advantage for the reactive routing protocols over proactive routing protocol is to reduce the bandwidth waste. When route discovery is less frequent than data transmission, reactive routing systems function effectively[26]. Route discovery is generally accomplished by flooding the network with route request packets. When route discovery is less frequent than data transmission, reactive routing systems function effectively.

3.2.2.1 Ad Hoc On-Demand Distance Vector Routing Protocol (AODV)

With low control overhead and route acquisition delay, Ad Hoc On-Demand Distance Vector Routing Protocol (AODV) creates efficient routes on demand. AODV is essentially a hybrid of

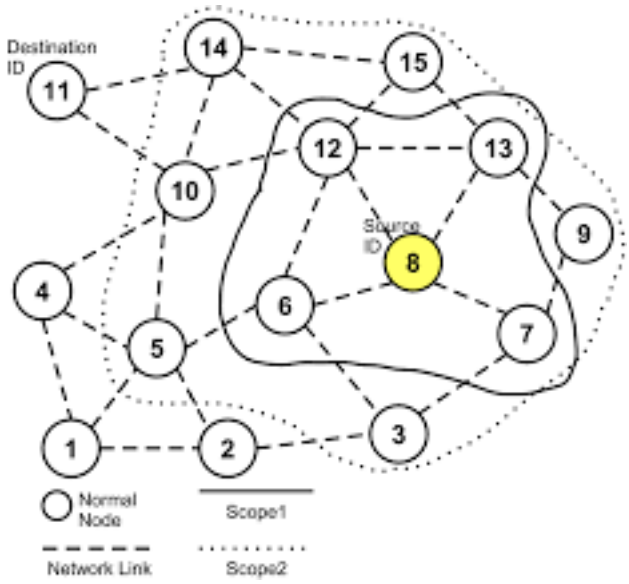

Figure 3.2: Fisheye hierarchical scopes

Dynamic Source Routing (DSR) and DSDV algorithms, using DSR's fundamental on-demand route discovery and maintenance mechanism as well as DSDV's usage of hop-by-hop routing sequence numbers to ensure loop-free routing. The source broadcasts a route request packet to determine a path from source to destination. The neighbors then broadcast the packet to their neighbors until it reaches an intermediary node with up-to-date route information about the destination or the destination itself. Because the route reply packet takes the same path as the route request packet, AODV only needs symmetric connections. The nodes along the path insert the forward route into their tables as the route reply packet travels back to the source. Figure 3.3 illustrate the AODV route discovery process. [6] [27]

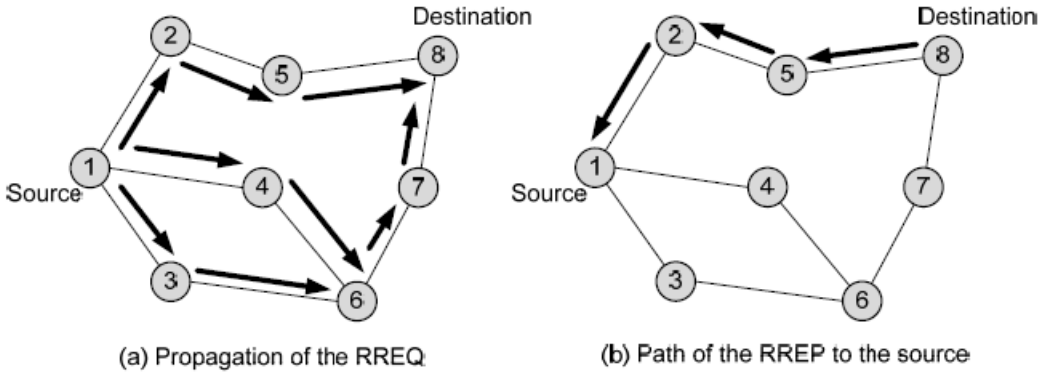

Figure 3.3: AODV route discovery

3.2.2.2 Dynamic Source Routing (DSR)

Dynamic Source Routing (DSR) offers a lot of advantages over routing protocols like AODV, and it may perform better on small networks. DSR has the benefit of allowing nodes to keep numerous routes in their route cache, allowing the node that wants to send a packet to some

destination to check its route cache for a valid route before beginning route discovery, and if one is discovered, route discovery is not required. However, DSR has a serious problem when dealing with huge networks, as its packets must carry the entire address (all hops in the path from source to destination). This indicates that this protocol will be ineffective on huge networks since the amount of overhead carried in each packet will grow as the network diameter grows, and it will consume a huge amount of bandwidth. Figure 3.4 illustrate how DSR packets look like. [31] [14]

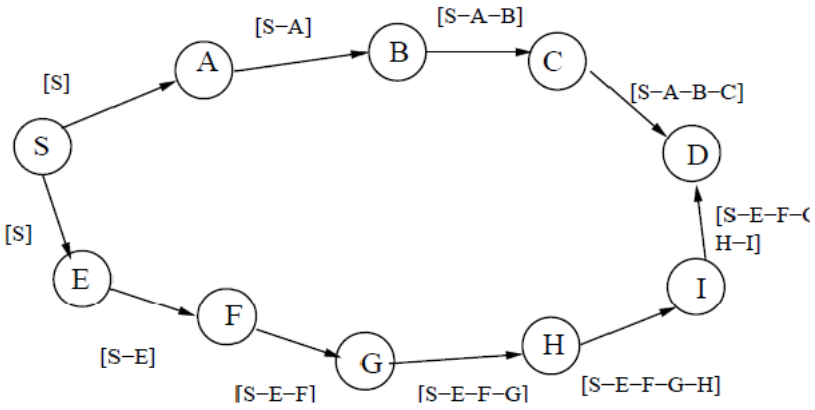

Figure 3.4: DSR packets

3.2.3 Hybrid Routing Protocols

Hybrid routing protocols are a new breed of protocol that is proactive as well as reactive. These protocols are intended to improve scalability by allowing nodes in proximity to collaborate to build a backbone that reduces route-finding overheads. The majority of this is accomplished by proactively maintaining routes to nearby nodes and finding routes to distant nodes using a route discovery strategy. The majority of hybrid protocols suggested to date are zone-based[4], which implies that each node sees the network as a series of zones. In the next subsection, the ZRP routing protocol will be discussed, while in the next section the ZHLS routing protocol will be discussed as our implemented approach.

3.2.3.1 Zone Routing Protocol (ZRP)

The nodes in ZRP each have a routing zone, which specifies a range (in hops) within which each node must maintain proactive network communication. As a result, routes are immediately available for nodes within the routing zone. Routes are determined on-demand for nodes outside the routing zone, and any on-demand routing protocol can be used to find a route to the needed destination. When compared to pure proactive protocols, this protocol has considerably reduced the amount of communication overhead. By allowing routes to be identified faster, it has also decreased the latency associated with pure reactive protocols like DSR. This is because, in order to identify a route to a node outside the routing zone, the routing simply needs to go to a node on the required destination's borders (routing zone's edge). The drawback of ZRP is that it can operate as a pure proactive protocol for high values of routing zone, but as a reactive protocol for small values. [3]

3.2.3.2 Zone Hierarchical Link state Routing (ZHLS)

Zone Hierarchical Link state Routing (ZHLS) split the whole network into zones that do not overlap. Each node in such a non-overlapping zone has a unique zone ID computed by GPS, and remember that each node in the whole network has a unique IP. The network's hierarchical structure (topology) is divided into two levels: node level topology and zone level topology, as shown in Figure 3.5. Intrazone and interzone routing tables are created by each node using the flooded interzone and interzone LSR packets. When a node wants to transfer data packets to a node in another zone, the source node broadcasts a zone level "destination zone discovery (DZD)" request to all neighbor zones, which need much less overhead than reactive protocols' flooding techniques. After obtaining the destination zone, the data packets may be forwarded to the needed destination zone and subsequently to the required target node using the interzone then intrazone routing tables.[13] [10] [2] [17]

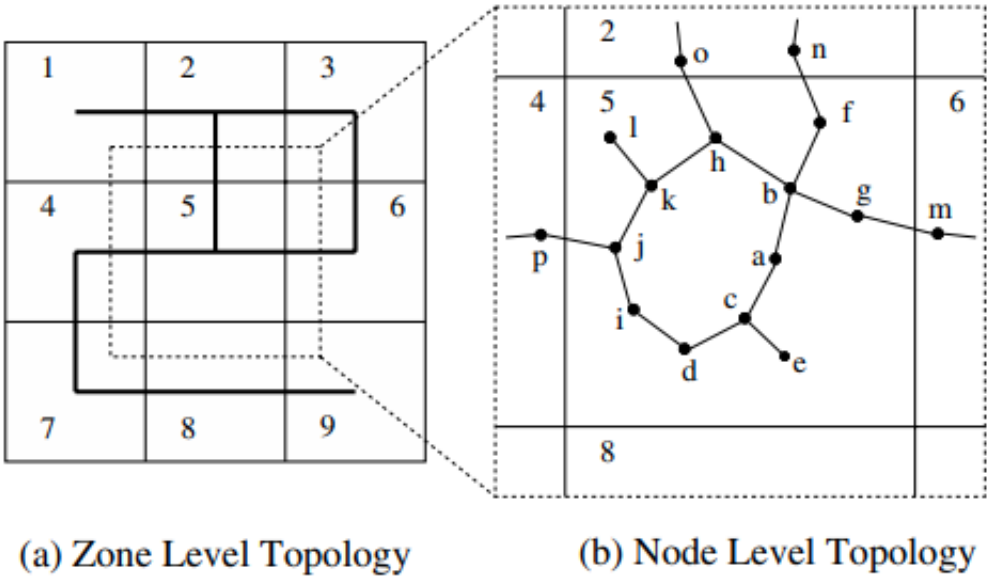

Figure 3.5: ZHLS network structure

3.2.4 Overall Comparison of All Unicast Routing Protocols

Summary comparison of unicast routing protocols can be shown in Table 3.1

3.3 Implemented Unicast Routing: ZHLS

From the previous comparison, we can deduce that, in completely proactive protocols, nodes constantly attempt to maintain routes to all other nodes. This implies they monitor all topology changes in real-time. When there are a lot of nodes with a high mobility rate, this might be challenging, and it will consume a lot of bandwidth. On the other side, in completely reactive protocols, nodes only receive routing information on demand, and paths are only built when nodes have data for a specific destination. These techniques cut routing

Routing Protocol Parameter	Proactive	Reactive	Hybrid
Storage Requirement	High	Low	Depends on size of each zone
Route Availability	Always available	Computed as per need	Depends on location of destination
Periodic Route Updates	Frequent	Sporadic	Used inside each zone
Delay	Low	High	Low for local destinations and high for interzone
Scalability	100 nodes	>100 nodes	>1000 nodes
Control Traffic	High	Low	Medium
Routing Information	Stored in table	Does not store	Depends on requirement
Routing Philosophy	Flat	Flat	Hierarchical

Table 3.1: Summary Comparison of Unicast Routing Protocols

costs significantly, but their performance is variable since they are never prepared for disruptive occurrences.

We determined that a hybrid routing approach that combines aspects of both proactive and reactive routing techniques would best meet our application needs in light of these constraints.

As stated above, Zone-Based Hierarchical Link State (ZHLS) is one of the best hybrid routing protocols for many reasons. The management of ZHLS nodes locations has been simplified. This is because the data transfer is not coordinated by a cluster head or a location manager. When compared to previous hybrid routing protocols, this implies there is no processing cost associated with cluster-head or Location Manager selection. This also implies that traffic bottlenecks and single points of failure may be avoided. When compared to pure reactive protocols like DSR and AODV, ZHLS offers lower communication overheads. Consider the scenario when a route to a remote destination is needed (i.e., interzone routing). The source node broadcasts a zone-level location request to all other zones, resulting in much less overhead when compared to reactive protocols' flooding approach. Another advantage of ZHLS is that the routing path may adapt to changing topologies because just the destination's node ID and zone ID are required for routing. As long as the destination does not relocate to another zone, no additional location searches are necessary.

The reasons we chose ZHLS as a unicast routing method are summarized in the following points:

- Simple location management as there is no cluster head or location manager node.
- It's ideal for tactical teams that are divided into groups and zones by nature.
- No traffic bottlenecks or single points of failure.
- Less memory requirements compared to proactive protocols.

- Lower control messages overhead using zone-level broadcasting compared to reactive protocols' flooding.
- Fewer computations when data is ready, so reduced battery usage as compared to reactive protocols.
- No location search is required as soon as the destination zone is known.

There is always room for optimization. ZHLS is one of the best unicast hybrid routing protocols but with some modifications, it can be better. The zone size is one of the most vexing aspects of any hybrid routing technology, particularly ZHLS. The size of a zone has a significant impact on network performance. There is always a better zone size than another, depending on the application and the mobility of the nodes. So we thought, why not make the zone size a variable that the user may set depending on the application? That was explained in details in system design section.

The second change was a technical one that affected the algorithm's implementation rather than its behavior. There will be nodes-level topology and zone-level topology, as well as intrazone and interzone forwarding tables, according to ZHLS. Consider the scenario when a route to a remote destination is needed. the source node have to apply the shortest path algorithm on both zone-level topology and node-level topology and look at both interzone and intrazone tables depending on the destination zone. With adding one layer of abstraction to both nodes and zones, we managed to encapsulate both node IP and zone ID in one class. So now there's only one topology that contain both nodes and zones, there is also one forwarding table for both. That modifications imply fewer computations and less memory and battery consumption.

In comparison to the previous method, the third and final update significantly improves router performance and saves a significant amount of bandwidth. As previously stated, ZHLS is a hybrid routing system, which implies it incorporates proactive routing approaches. The periodic changing of Interzone LSR control messages, which is used to convey zone information throughout the network, is one of these characteristics. The original technique required each node in each zone to flood this message, but just one node from each zone is enough because all nodes in the same zone will flood the same data. So, using some criteria, we were able to assign this work to only one node in each zone, greatly reducing control message overhead and improving system efficiency.

3.4 Multicast Routing in MANETs

Multicast routing allows you to support group-oriented applications while using less bandwidth. Because of the growing demand for such applications, as well as the inherent characteristics of MANETs such as lack of infrastructure and node mobility, secure multicast routing has become a critical yet difficult topic.

Multicast is an essential communication pattern that involves the sending of packets among a group of two or more hosts, and therefore is suitable for group-oriented computing. The use of multicasting in MANETs has several benefits. For example, it can lower the cost of communication and increase the accuracy of the wireless channel, while transmitting numerous copies of the same data by exploiting the inherent broadcasting characteristics of wireless transmission. Instead of delivering data via multiple unicast connections, multicasting lowers channel capacity usage, reduces the cost of the sender and router's processing and energy, and communication latency. [15] [16]

3.4.1 Classification Of Multicast Routing Protocol

Figure 3.6 demonstrates the dependency between the different and the main classification aspects for multicast routing protocol, including multicast topology, initialization approach, routing scheme, and maintenance approach.

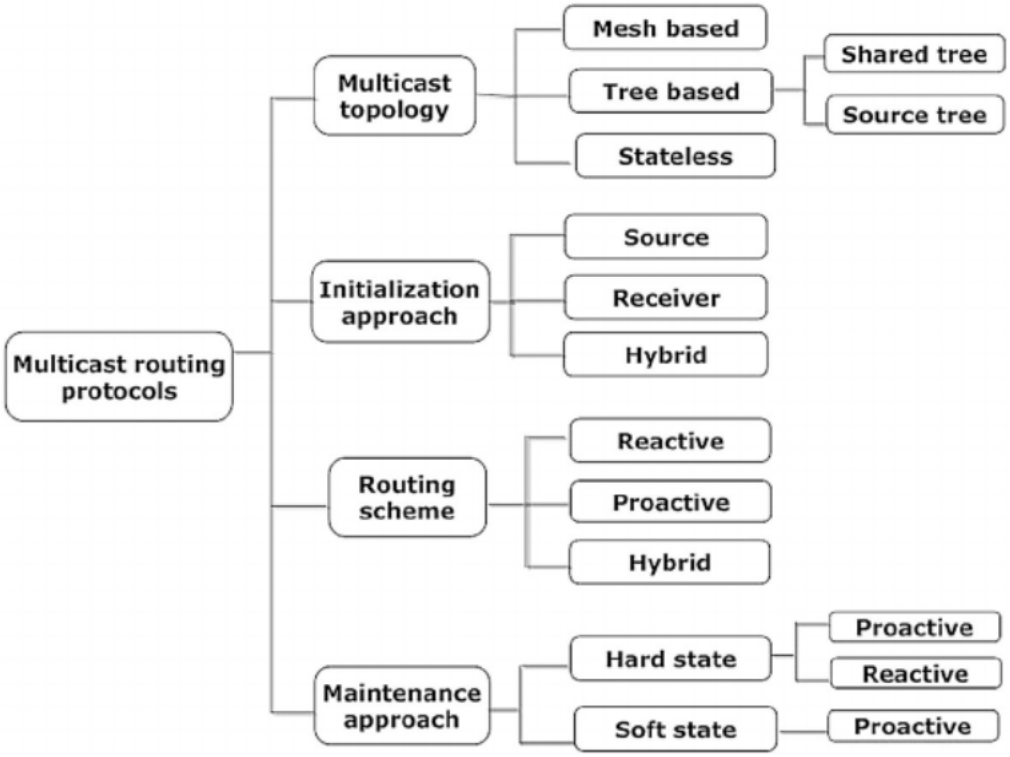

Figure 3.6: classification of multicast routing protocols

3.4.1.1 Multicast Topology

Multicast topology has three approaches which are tree-based, mesh-based, and stateless approach. These three methods are described as follows:

1. Mesh-based approach

   Mesh-based multicast protocols could have many routes between the same source and receiver pair. Current studies demonstrate that tree-based protocols are not always the best fit for multicast in a MANET when network topology changes often. In such an environment, mesh-based protocols tend to outperform tree-based approaches given the availability of different routes, which allows multicast datagram packets to be sent to the receivers even if connections fail. In this method, numerous routes are built in the whole network. These redundant routes are essential in link failure situation and give better packet delivery ratio

2. Tree-based approach

   Most systems for providing multicast in wired networks seem to be either source-tree-based or shared-tree-based. A unique route between source and receiver exists. This path and other paths are controlled by a general purpose node named core-node.

   There are two main types of tree-based approach, which are described as follows:

- first is source-tree-based, whereby each source establishes a separated tree that includes the source node as the root node and then all receivers exist under this node.
- second is shared-tree-based, in which one tree is created in the overall system, which contains all sources and receivers, so in this case, a core node controls the tree and behaves as a root to the tree.

3. Stateless approach

   The stateless multicast approach, in which the source node explicitly specifies the list of targets in the packet header, is suggested to reduce the impact of such issues in tree-based and mesh-based approaches. Stateless multicast focuses on specific group multicast and expects the underlying routing protocol would take care of delivering the packet to appropriate destinations based on the addresses provided in the header.

3.4.1.2 Routing Initialization Approach

Routing initialization is classified into three methods which are source-initiated, receiver-initiated, and hybrid approach. The three methods are described as follows:

1. Source-initiated approach

   It is a method in which the multicast group creation and maintenance functions are handled by the source node. In order to start a new multicast group, the source node broadcasts a join query message all across the network, and any node wishing to join that multicast group responds by a join reply message.

2. Receiver-initiated approach

   It is a method in which the receiver node looks for a dedicated source inside the multicast group. The receiver node broadcasts a join query message across the network to enter a new multicast group, and thus the source node or core node responds by a join reply message containing the multicast group core route.

3. Hybrid approach

   The multicast group creation and maintenance functions are handled by either the source node or the receiver node in this method, which integrates specific characteristics from the source-initiated and receiver-initiated approaches.

3.4.1.3 Routing Scheme

Routing scheme is classified into three methods which are table-driven (proactive), on-demand (reactive), and hybrid approach. The three methods are described as follows:

1. Table-Driven Scheme (Proactive)

   In a network using a proactive routing protocol, every node keeps one or more tables describing the complete network topology. These tables are frequently updated in order to keep up-to-date routing information of each node to any other node. To keep routing information up-to-date, topological information has to be transmitted between the nodes on a frequent basis, resulting in relatively significant overhead on the network. But on the other hand, routes will always be accessible on request.

2. On-demand scheme (Reactive)

   It attempts to build up routes on-demand, if a node wishes to start communication with another node to which it has no path, the routing protocol would seek to construct such a path. Reactive multicast routing protocol offers higher scalability than proactive multicast routing protocol. Unfortunately, while using reactive multicast routing protocol, source nodes may suffer from extra delay for path finding before they may send data packets.

3. Hybrid scheme

   It integrates the proactive and reactive methods in one strategy, in order to overcome the limits of both protocols and enhance their benefits. An example of a hybrid approach is zone routing protocol, which stores routing tables for a local zone, and creates routes on demand for targets outside this local neighborhood. It restricts the range of the local zone by describing a maximum hop number for the local zone.

3.4.1.4 Multicast Maintenance Approaches

Multicast maintenance is classified into two methods which are soft-state, and hard-state approaches. The two methods are described as follows:

1. Soft-state approach

   This is a method in which broken link repair operation is started regularly by flooding the network with continuous control packets to seek alternative paths between source and destination. This method has the benefit of reliability and improved packet delivery ratios, but it creates considerable overhead over the network as it repeatedly floods the network with control packets.

2. Hard-state approach

   This is a method in which broken links maintenance procedure is created by two kinds, including reactive and proactive. In reactive methods, broken link recovery procedure is started only when a connection breaks. The second kind is proactive strategy, in which routes are reconfigured before a connection breaks, and this may be accomplished by using local prediction methods based on GPS or signal strength.

3.5 Comparative study of Multicasting

In this section we compare some commonly used multicast routing protocols and compare them and choose the best fit for our application, the comparative study done on the following protocols:

- On Demand Multicast Routing Protocol (ODMRP)
- Protocol for Unified Multicasting through Announcement (PUMA)
- Multicast Ad-hoc On demand Distance Vector Protocol (MAODV)
- Overlay Boruvka-based Ad-hoc Multicast Protocol (OBAMP)
- Application Layer Multicast Algorithm (ALMA)
- Enhanced version of ALMA (ALMA-H)

3.5.1 Multicast routing protocols

- On Demand Multicast Routing Protocol (ODMRP):

  ODMRP [18] is a reactive protocol which is mesh-based and source initiated protocol, it also maintains a mesh by following the soft-state approach.

- Protocol for Unified Multicasting through Announcement (PUMA):

  PUMA [21] is reactive protocol and distributed, it is mesh-based and receiver initiated protocol, PUMA's all transmissions are broadcast and doesn't depend on any unicast protocols.

- Multicast Ad-hoc On demand Distance Vector Protocol (MAODV):

  MAODV [7] is a reactive tree based protocol and hard-state, it uses a broadcast route discovery mechanism to discover the multicast routes on demand, it's a multicast extension of the AODV protocol.[**chow2008multiple**]

- Overlay Boruvka-based Ad-hoc Multicast Protocol (OBAMP):

  OBAMP [24] is a proactive mesh-first overlay multicast protocol, it is used to find the minimum spanning tree protocol which is a receiver initiated and soft-state approach.

- Application Layer Multicast Algorithm (ALMA):

  ALMA [11] is a proactive tree-based protocol, it is a receiver initiated protocol and soft-state approach, it is a flexible and highly adaptive overlay multicast protocol.

- Enhanced version of ALMA (ALMA-H):

  ALMA-H [11] is a proactive tree-based protocol, it is an enhanced version of ALMA for tree efficiency, it is a receiver driven and soft-state approach.

3.5.2 Performance Evaluation Of Multicast Routing Protocols

The performance evaluation criteria used to compare these protocols are Throughput, Reliability, End-to-End delay and Packet Delivery Ratio by increasing the numbers and speed of the nodes. [29]

1. Throughput: the ratio between the number of data packets generated to the number of the data packets received.
2. Reliability: the ratio of the successful end-to-end data delivery.
3. End-to-end delay: the interval elapses between time of sending a packet and time of successfully delivering it.
4. Packet Delivery Ratio: the ratio between the number of data packets delivered to the destination to the number of packets generated at the source.

3.5.2.1 Scenario is varying the number of nodes

In this scenario the performance is measured for the four performance metrics by increasing the number of nodes from 20 to 80 with fixed speed of 0 m/s, the following graphs show the comparison between the performance of the six protocols.

Figure 3.7 top left graph shows the simulation results of throughput in (Kbps) versus number

of nodes.
The results show that on increasing the number of nodes, ODMRP [18] gives higher throughput than the other protocols that is because ODMRP deliver data packets in high rate as its operation is on-demand, on the other hand MAODV has the worst throughput as most of the nodes don't participate in data transfer

Figure 3.7 top right graph shows the simulation results of reliability versus the number of nodes where the number of senders is only one, so all protocols demonstrated high reliability. ODMRP and PUMA achieve the highest reliability as the number of nodes increases, the network becomes strongly connected and improves reliability.

Figure 3.7 bottom left graph shows the end-to-end delay in (sec) versus the number of nodes.
ODMRP shows the best delay performance than the other protocols because its route discovery mechanism is fast.

Figure 3.7 bottom right graph shows the results of packet delivery ratio versus the number of nodes.
ODMRP has a higher packet delivery ratio than the other protocols, PUMA and MAODV are pure on-demand protocols, however ODMRP is dynamic on-demand protocol.

3.5.2.2 Varying the speed of nodes

In this scenario the performance of the six protocols are measured for the four performance metrics by increasing the speed of nodes from 0 to 20 m/s with fixed number of nodes 80, the following graphs show the comparison between the performance of the six protocols.

Figure 3.8 top left graph shows the simulation results of throughput versus the mobility speed in (m/s).
We can see that the throughput decreases for all the six protocols as the mobility speed increases, and ODMRP has the best throughput as finding the route needs more routing traffic when speed increases. So less of the channel will be used to transfer data.

Figure 3.8 top right graph shows the simulation results of reliability versus the mobility speed.
We can see that by increasing the mobility speed ODMRP and ALMA-H have better reliability that the other protocols. This is because they have less transmission delay of messages.

Figure 3.8 bottom left graph shows the simulation results of end-to-end delay versus mobility speed.
When increasing the speed of the nodes, the topology changes frequently and then the probability of link breaking will increase. So additional route recovery process and route discovery process may be required. So as the speed increases, the end-to-end delay will also increase.

Figure 3.8 bottom right graph shows the simulation results of packet delivery ratio versus mobility speed.
The packet delivery ratio of all the protocols decreases when the speed increases, and ODMRP has better results than the other protocols.

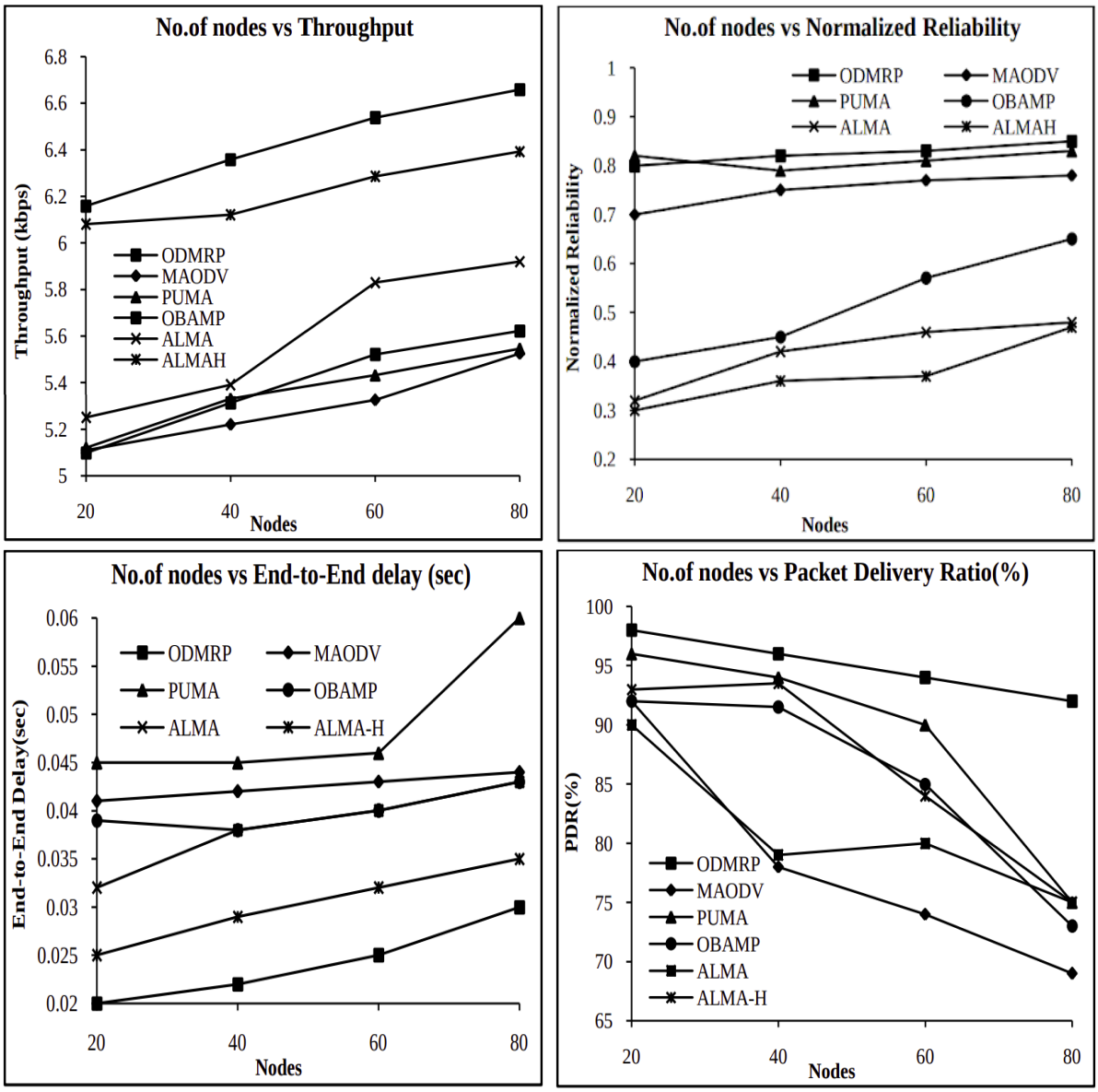

Figure 3.7: Changing number of nodes vs reliability, throughput, delay and packet drop ratio

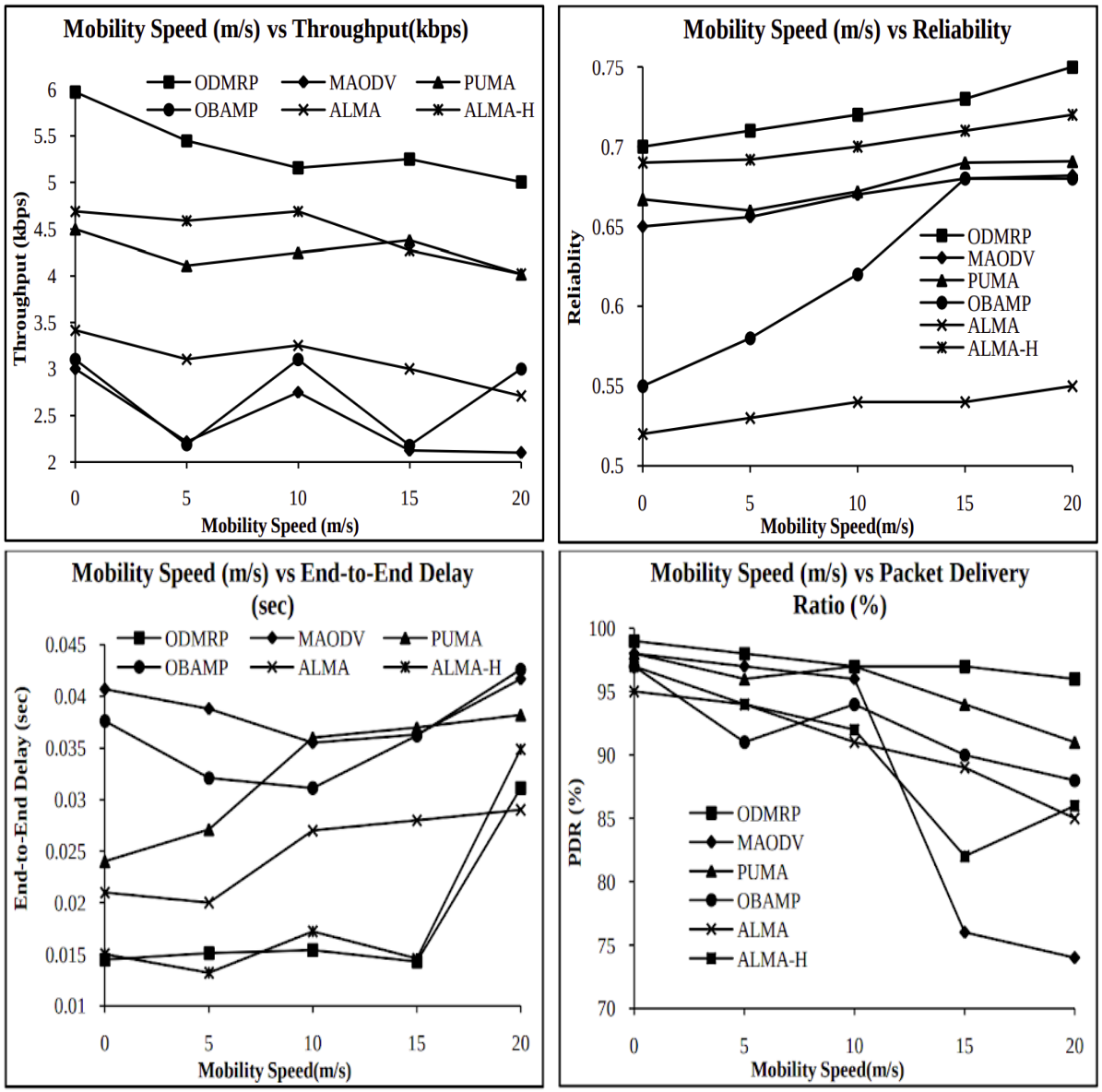

Figure 3.8: Changing speed of nodes vs reliability, throughput, delay and packet drop ratio

3.6 Implemented Multicast Routing: ODMRP

The Wireless Adaptive Mobility (WAM) Laboratory at the University of California, Los Angeles created the On Demand Multicast Routing Protocol (ODMRP) [18], to reduce channel overhead and enhance scalability, ODMRP employs on-demand routing methods. It constructs a forwarding mesh for each multicast group using the idea of a forwarding group, which is a collection of nodes responsible for passing multicast data. The disadvantages of multicast trees in mobile wireless networks, such as inconsistent connectivity, traffic concentration, frequent tree reconfiguration, and non-shortest path in a shared tree, are addressed by maintaining and utilizing a mesh instead of a tree. To keep multicast group members, a soft-state technique is used. To quit the group, no specific control message is necessary. We feel that ODMRP is more appealing in mobile wireless networks because of the reduced channel/storage overhead and the relaxed connection. The advantages of ODMRP are highlighted by the features listed below.

3.6.1 ODMRP Overview

ODMRP is a mesh-based multicast scheme that employs a forwarding group idea to ensure that only a subset of nodes forwards multicast packets, rather than a traditional tree-based multicast scheme, It uses on-demand processes to create routes and manage multicast group membership dynamically. ODMRP is ideally suited for ad hoc wireless networks with mobile hosts in which bandwidth is restricted, topology changes often and quickly, and power is limited.

The Query and Reply stages make up the ODMRP. A packet Join Query is broadcast by source on a regular basis. Every node in the network saves the upstream node address in the route database and rebroadcasts the packet to its neighbor nodes when it receives a non-duplicate Join Query packet via the reverse path technique. When a multicast receiver receives a Join Query packet, it generates and transmits a Join Reply to its neighbors.
Following the learned reverse path, this Join Reply packet is sent all the way back to the source, and the nodes on the reverse path become the forwarding group. The following is how data is provided by the forwarding group nodes: The source broadcasts data packets, while nodes in the forwarding group rebroadcast non-duplicate packets so that packets can be transmitted to recipients. The forwarding group is a collection of nodes that are in charge of transmitting multicast data, thus establishing a mesh between all senders and receivers. When a new node wishes to become a receiver in a multicast group, it waits for the next Join Query and replies with a Join Reply packet. If a receiver becomes detached from the forwarding mesh owing to mobility, it must also wait for the next Join Query flooding.
Without a local route recovery mechanism, ODMRP's frequent route refresh and redundant forwarding maintain a high packet delivery ratio. A soft-state technique is used in ODMRP to keep track of multicast group members; no explicit control messages are needed to join or exit the group. A source node simply stops delivering a Join Query when it has no more packets to send. If a receiver wishes to leave a multicast group, it does not answer to the multicast group's Join Query.
If a forwarding node does not get the Join Reply after reaching a timeout that is a multiple of the refresh interval, it becomes a non-forwarder.

After comparison to the previous routing protocols and other protocols using the four per-

formance metrics: throughput, reliability, end-to-end delay and packet delivery ratio under two conditions: increasing the number of nodes and increasing the speed of the nodes. We found that In all cases, ODMRP outperforms the other multicast routing protocols, leading to the conclusion that ODMRP is effective and efficient in highly dynamic conditions and scalable to numerous multicast nodes, as for route discovery and maintenance, ODMRP uses periodic network-wide flooding. The purpose of this design is to guarantee robustness in the face of mobility and poor wireless connection propagation, and this feature would be very useful especially in rugged environments.

The reasons we chose ODMRP as our multicast routing protocol are summarized in the following points:

- The ability to route unicast traffic.
- Low overhead in terms of channels and storage.
- Use of the most recent shortest routes.
- The ability to withstand host movement.
- Reliable route and forwarding group development.
- Wireless settings' broadcast nature is used.
- Maintenance and exploitation of multiple redundant paths.
- Scalability through the use of effective flooding.
- The protocol's extensibility.
- Limited power usage.

There is always room for optimization. ODMRP is one of the best multicast routing protocols but with some modifications as We did in broadcasting the join reply packets by using the filled tables to only sending the join reply packet to the upstream nodes, it also can be optimized further by other extendable versions of ODMRP:

- E-ODMRP version of ODMRP with adaptive refresh. Rather than mobility prediction, receivers report connection breakages, which drives adaptation. A basic local recovery is easily incorporated with the adaptive refreshing method. In light load, EODMRP [22] may have a slightly lower packet delivery ratio than ODMRP, but it may be better in high loads.
- RODMRP [23] which seeks to be robust to network or node failures and to provide a continuous multicast service for live data streaming.
- It also could be extended to work with zones.

Chapter 4

System Design and Architecture

This chapter describes the following in detail:

- All the system modules.
- The flow of data between modules.
- Modules' functionalities.
- Design constraints.
- The decisions we took about the modules' functionalities.

The overall system could be broken down to mainly 3 components:

- The Router: Implements unicast, multicast and broadcast within a control plane and forwarding plane.
- Unit & Command Center Client (CMD): 2 applications for tactical teams communications.
- Testbed: Set of scripts, kernel modules, programs and Graphical User Interface (GUI) for system testing and development which emulates the Hardware (HW) and simulates the environment in which the system will be deployed.

This chapter discusses the first 2 modules, while the testbed is discussed completely in chapter 5, because it's not part of the final deployed system, but rather built for testing purposes.

4.1 Overview and Assumptions

Units are soldiers who are deployed in the operations field, while commanders are in their vehicles or buildings sending commands to the soldiers/officers and monitoring their locations and watching streaming and audio recording from soldiers and sending and receiving code messages to the deployed soldiers.

Commanders have powerful devices, while units have low-end devices with low range for maximum battery conservation.

4.1.1 Reliability

The following must be delivered reliably (with guarantee of delivery):

- Code messages.
- Audio messages.

The following can be delivered unreliably (*no* guarantee of delivery):

- Video streams.
- Position and heartbeat messages (minimum 80% delivery success rate).

4.1.2 Speed

The system allows nodes to communicate with low latency and high throughput.

4.1.3 Security

- All transmitted data are encrypted. Including the headers.
- Authentication is required for accessing command center by its UI.
- Units don't persist any data, messages self-destruct after receiving and playing them.

4.2 System Architecture

The system is composed of devices (*nodes*) running Linux-based operating systems and have certain programs running in them.

4.2.1 Nodes

All nodes are provided with wireless communication modules that follow `IEEE 802.11` standards for wireless communications.

There are 2 types of nodes:

1. Units.
2. Command Centers (shortened as *CMD.*)

4.2.1.1 Units

Units (software) run on devices with deployed units (persons) in the operation field, the single device has:

- LCD screen, to show the code messages and device state (battery and location and number of received audio messages.)
- Helmet video camera. Could be mounted on a vehicle if the device runs on one.
- Audio input from microphone with button to start recording.
- Audio output, to hear received recorded messages.
- Keypad, to enter code messages.
- GPS (or any other position detection system.)

- One heartbeat sensor, that streams the beats per minutes of the person who operates the device.

The unit device has the following features:

- Low power consumption.
- Running on battery.
- Low wireless range.
- High mobility.
- Operated by one person.

A unit device has 2 programs:

- Router: Implements routing protocol.
- Unit Client Daemon: Connected to device hardware and network interface and provide all unit features.

4.2.1.2 Command Centers

High-end computers at the command and control centers, accessed by units leaders.

The command center device has the following features:

- Capable of high power consumption.
- Powerful CPUs.
- Big storage and RAM.
- Operated by multiple people with multiple wide screens.
- Wide wireless range.
- Installed nearby the operation field, and has a connection to devices in the field.
- Low (or zero) mobility.

A command center computer has 3 programs:

- Router: Implements routing protocol, same router as in unit devices.
- Command Client Daemon: Exposes an interface to UI program, connects to units clients and handles all communication with units.
- Command Client UI: Connects to Command Client Daemon, shows all data in the daemon and controls it.

4.2.2 Block Diagram

Figure 4.1 shows the overall C4IAN system modules.

Unit or Command Center are applications that communicate directly with the TCP/UDP standard OS interface. You would only use unit or command center on one device. They are not aware of the router that sits on the bottom, this means they should work on localhost, which we already do for testing purposes to isolate system modules.

The router sits in the bottom, it's backward compatible with the Linux kernel network stack design, this means that programs in the application layer won't change when using the

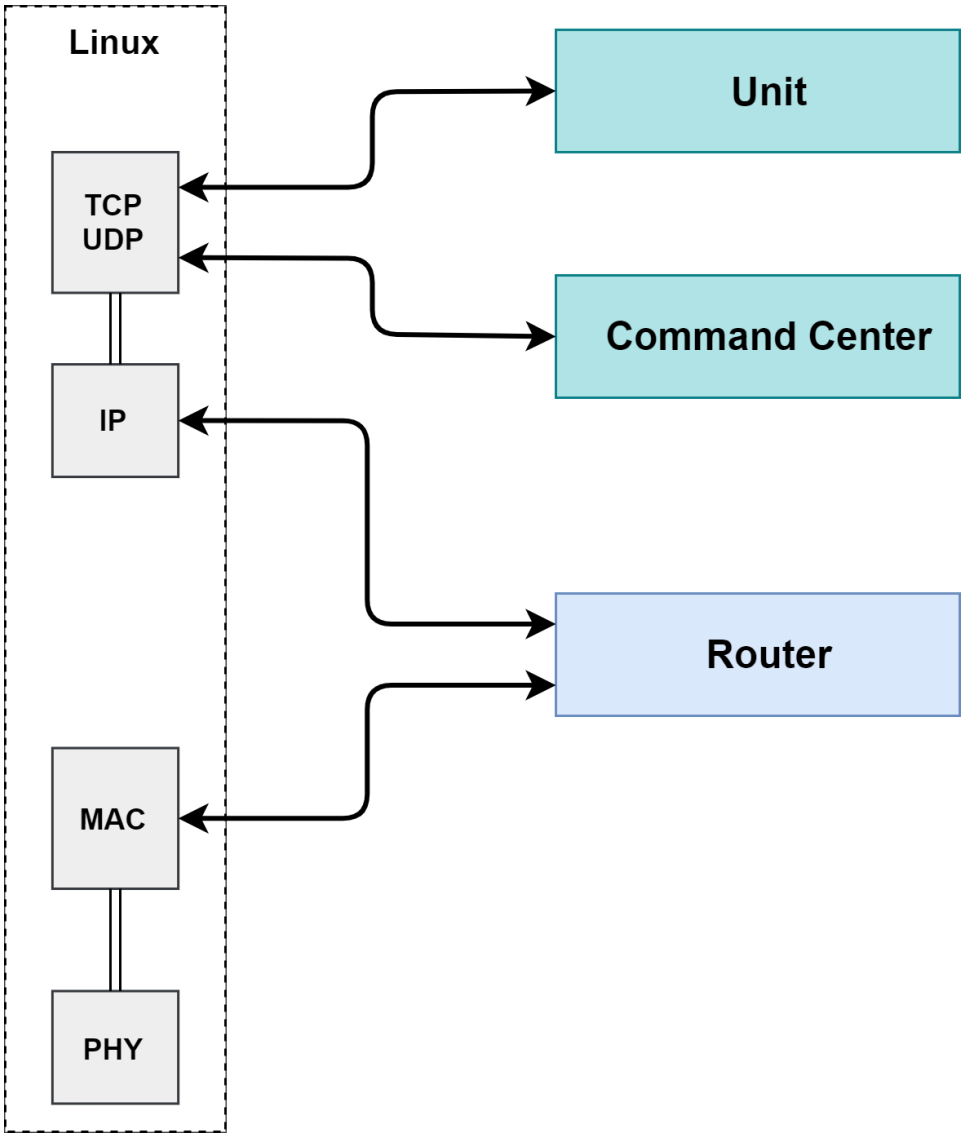

Figure 4.1: Overall System

router. The router takes packets going out of the system, and encrypts the packet and wraps it with our own headers and decides the forwarding of the packet. Then the router puts back the packet in the MAC layer, so it's sent to the destination.

In the testing environment, as we will detail in chapter 5, we will isolate routers in each container with its own network stack, and create either a unit or command center program depending on the node type.

4.3 The Router

Routing is the process of filling the forwarding tables, while forwarding is the process of taking a packet and deciding the destination from the forwarding table. The router is the main component of the project, the rest are applications that show the effectiveness of the router. The router is backward compatible with Linux's standard network stack and works completely in the user space.

The router enables all programs and kernel modules that communicate with the IP layer

to run over an ad-hoc network transparently without changing any line of code.

4.3.1 Functional Description

The router takes the packet after wrapping the TCP/UDP/ICMP with IP headers, and wraps it with its packet Zone IDentification Protocol (ZID) for interzone forwarding and encrypts the whole packet. The router also encrypts all packets including control packets before them leaving the device. This stops any device that doesn't share the same credentials from participating in the network and ensures the security of the communications.

The router exchanges control packets to build the topology that may be used for forwarding when a packet is available.

The router provides unicast, multicast and broadcast services to the upper layers based on IP address patterns.

4.3.2 Modular Decomposition

Figure 4.2 shows the modules of the router in detail.

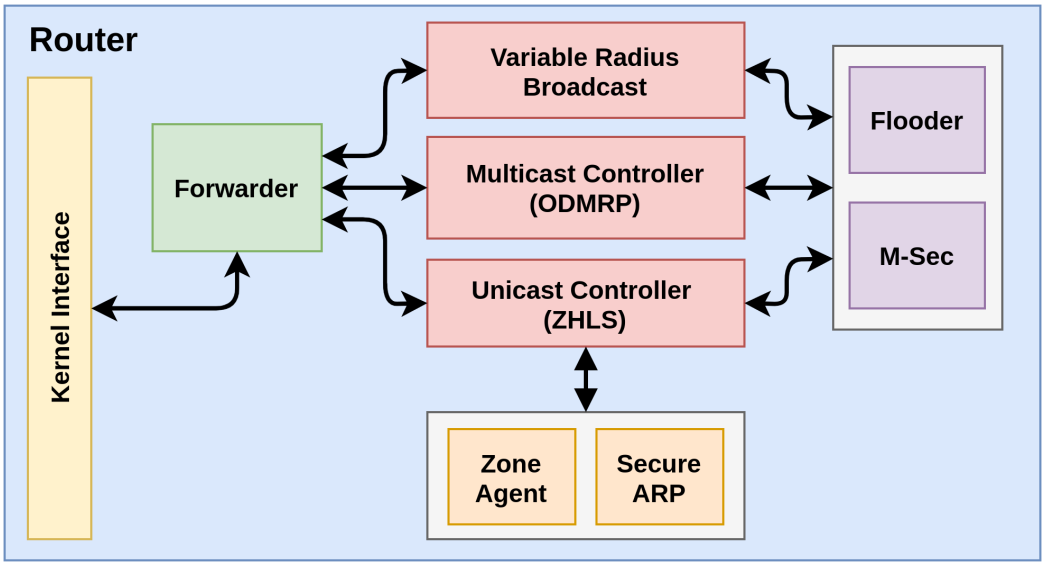

Figure 4.2: The Router Modules

Broadcast routing implements a new protocol we introduced called "Variable Radius Broadcast" (VRB), which limits the broadcast to a given geographical radius. Unicast routing implements the "Zone Hierarchical Link State Routing" (ZHLS) protocol, which solves the scalability requirements for the routing. Multicast routing implements the infamous protocol "On Demand Multicast Routing" (ODMRP) protocol.

All the three routing protocols are implemented in the control plane. They fill different kinds of forwarding tables suited for each protocol.

In the forwarding (data) plane there is the forwarder which is a complex module that takes packets leaving and entering the device and either forwards them or injects them back based on the 3 protocols' forwarding tables and has channels with the control plane to kick reactive routing decisions, like finding a zone for destination or starting the process of sending join query messages for ODMRP.

4.3.2.1 Kernel Interface

The router reads and writes packets from the kernel in 3 ways:

- Netfilter Queue: Used to steal packets from the kernel. They were designed in mind for firewall and deep-packet-inspections applications, but it suited our use case. The kernel gives us each packet leaving the device, and we drop them and deal with them ourselves and send it to the device driver.
- Packet Sockets: Used to read/write packets at the device driver, writing to it send a packet out the device.
- Raw Sockets: Used to inject packets into the IP layer after receiving them from the

4.3.2.2 Forwarder

The forwarder constitutes our data plane. It is where all the data packets pass and get routed. The forwarder listens to packets coming from the IP layer and packets coming from the MAC layer. Packets coming from the IP layer are packets originating at the current node, while packets coming from the MAC layer are packets arriving from other nodes, either addressed to the current node or to another destination. The forwarder is responsible for controlling the flow of those data packets. Figure 4.3 shows how the data flows through the forwarder.

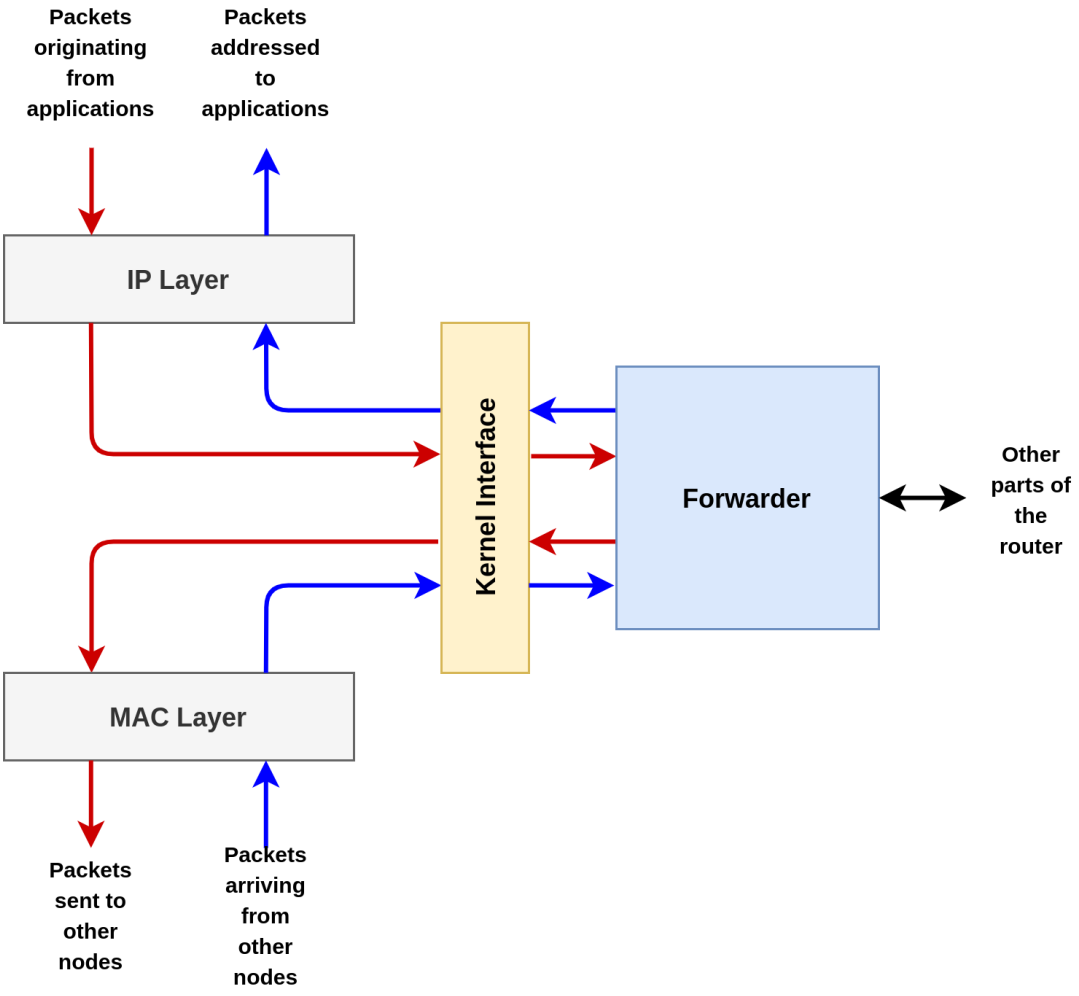

Figure 4.3: Data Flow Through the Forwarder

When the forwarder receives data from the IP layer, it first checks the type of routing required for the IP packet depending on the destination IP address, which can be unicasting, multicasting, or broadcasting:

- **Unicasting**
  - The destination zone is found through the unicast controller.
  - A ZID header is constructed and prepended to the packet.
  - The next hop is determined through the unicast forwarding table constructed by the unicast controller.
  - The packet is encrypted and send to the next hop through the MAC layer.
- **Multicasting**
  - The next hop is determined through the multicast forwarding table constructed by the multicast controller.
  - The packet is encrypted and send to the next hop through the MAC layer.
- **Broadcasting**
  - The packet is encrypted and sent through the variable radius broadcast handler.

The forwarder uses different listeners to receive unicast, multicast, and broadcast packets from the MAC layer. The packets are differentiated using a unique ethertype used by the forwarder for each packet type.

When the forwarder receives a unicast packet from the MAC layer, it decodes, decrypts and validates the ZID and IP headers. The forwarder then checks whether the current node is the destination of the packet. If so, the packet is sent to the IP layer. Otherwise, the forwarder checks if the destination node lies within the current zone's node through the ZID header. If it does, the forwarder checks the unicast forwarding table for the next hop for the destination node. Otherwise, the forwarder checks the unicast forwarding table for the next hope for the destination zone. The IP header is then reconstructed to update the TTL and checksum, and the packet is forwarded to the next hop though the MAC layer. An interesting case is when a packet reaches its destination zone, but the forwarder cannot find the destination node in its zone. This may occur due to the mobility of the nodes, as the node may have left the zone after the packet was sent from the source. In this case, the forwarder uses the unicast controller to find the new zone of the destination, and reconstructs the ZID header with the new destination zone.

When the forwarder receives a multicast packet from the MAC layer, it decodes, decrypts and validates the IP header. The forwarder then checks to see whether the current node is a member of the destination multicast group. If so, the packet is sent to the IP layer. Regardless of whether the current node is a member, the packet may need to get forwarded according to the structure of the multicast mesh. The forwarder checks if it should forward the packet using the multicast forwarding table. If it should, it reconstructs the IP header to update the TTL and checksum, and the packet is forwarder to the next hop.

4.3.2.3 Variable Radius Broadcast (VRB)

Research in ad-hoc routing gave most of its attention to unicast protocols, while broadcast and multicast were left in the dark. Broadcasting in infrastructure networks uses the subnetting

mechanism to limit the broadcasting and put boundaries, while this is most effective to decide who participates in broadcasting, but this is not applicable to ad-hoc networks because there are no subnets otherwise moving devices will be assigned different IPs each time they change cluster and this will disrupt applications logic and break the backward compatibility.

Given that ZHLS already requires geographical locations to be maintained and shared, we figured out a way to put boundaries on the broadcasting using the current location of each device.

The protocol is simple, you know which zone you are in, when some device wants to broadcast a message it will put the broadcasting radius in the lowest byte of the destination IP while setting the rest of the bits to 1. when a device receives a broadcast a message it will check the src zone and the radius and its zone and calculates the radius between its zone and the src zone, if the calculated radius is bigger than the given radius (in the destination IP) it won't forward the packet anymore otherwise it floods the message.

4.3.2.4 Flooder

Flooder is a component of any routing system that distributes every incoming packet via every outbound connection except the one from which it originated. Because C4IAN uses wireless channels as a physical layer to suit the nature of tactical teams on battlefields, and because wireless networks are broadcast by their inherent nature, this implies two important aspects in our flooding approach implementation:

- There is no need to figure out which outgoing connection the incoming packet should be sent to; simply broadcast it.
- The need for a flood control method in order to avoid transmitting duplicate packets and endless recycling of the same packet.

Because C4IAN uses a zone-based routing approach for unicasting, we needed to manage the flooding area, therefore we built two flooders: a zone flooder and a global flooder. The zone flooder is primarily used to flood intrazone LSR control messages, whereas the global flooder was used by the unicast algorithm to flood interzone LSR control messages, and it was also used by the multicast algorithm.

Before flooding any packet, it must contain a ZID header (not required in global flooder) and a flooding header that is shown by Figure 4.4.

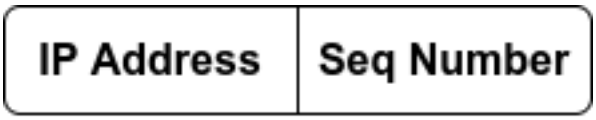

Figure 4.4: Flooding Header

The header fields are as follows:

- **IP address:** The IP address of the sender.
- **Seq Number:** The sequence number of that packet from that source.

Seq. number is used to control the packet flooding in the network and to prevent recirculation of the same packet indefinitely. So for any router to accept an incoming packet and flood it to all neighbors, it must satisfy three conditions:

- It must not be flooded by the receipt router itself.

- It must be flooded by a router that exists in the receipt router zone.
- It must have a sequence number that is higher than the latest one stored for that source.

It is obvious that the second condition is only required by the zone flooder.

According to the preceding definition, each router must keep a sequence number for each other router in the zone (or the network in the case of global flooding) that has the value of the sequence number attached with the most recent packet from that source. Each entry in the preceding flooding table has a timer connected to it in addition to the sequence number. Once the timer goes off, the entry must be deleted immediately. This timer will allow accepting packets from a router that has gone down for whatever reason and reset its sequence number to zero.

4.3.2.5 M-Sec

M-Sec stands for "MANET Security" which is the encryption layer we introduced in the router. It encrypts all outcoming packets (including their headers) and decrypts all incoming packets.

M-Sec uses AES256-CFB for its speed and no found vulnerabilities yet. We didn't choose asymmetric encryption like that used in IPsec because of the problem of certificates sharing and signing, which introduces unwanted centralization which weakens the security of the whole system in case the central authority was compromised. Also, central authorities add to the complexity of the system, which makes it harder for huge organizations like the military and any tactical teams in general to adapt to.

M-Sec uses passphrase for its key derivation, which is done using pbkdf2 with 4096 iterations and a fixed salt. The reason to use passphrase is to integrate seamlessly with the military way of securing communications, which they have been trained to for decades. This enables C4IAN to be a drop-down replacement for traditional radio systems without more training.

When we encrypt a packet, we encrypt each header on its own, this enables quick decryption for the needed header to decide its destination in the forwarding plane and in case we need to update the header's data like TTL and needed to recalculate the checksum, no need to encrypt the whole packet.

Using CFB mode enables the packet to be encrypted without increasing its size because it's a stream mode. It was also chosen among other stream modes because of its capability to detect tampering with the packet, in case someone changed one bit the whole stream will change completely which will be detected easily either by the IP header checksum or version or TCP/UDP/ICMP length or checksum or the application layer. This provides authentication besides confidentiality.

4.4 Unicast Controller

4.4.1 Functional Description

The unicast controller is in charge of one-to-one transmission between two network nodes; that is, one sender and one receiver, each with an IP address. At every node, it creates a forwarding table, which includes the best next-hop to forward a packet to for each possible destination. Collectively, this constitutes finding a path from every possible source to every

reachable destination. As indicated in the preceding section, we believe Zone Hierarchical Link state Routing (ZHLS) is one of the best hybrid unicast protocols for this task. As a result, we chose it as our unicast routing protocol.

4.4.2 Modular Decomposition

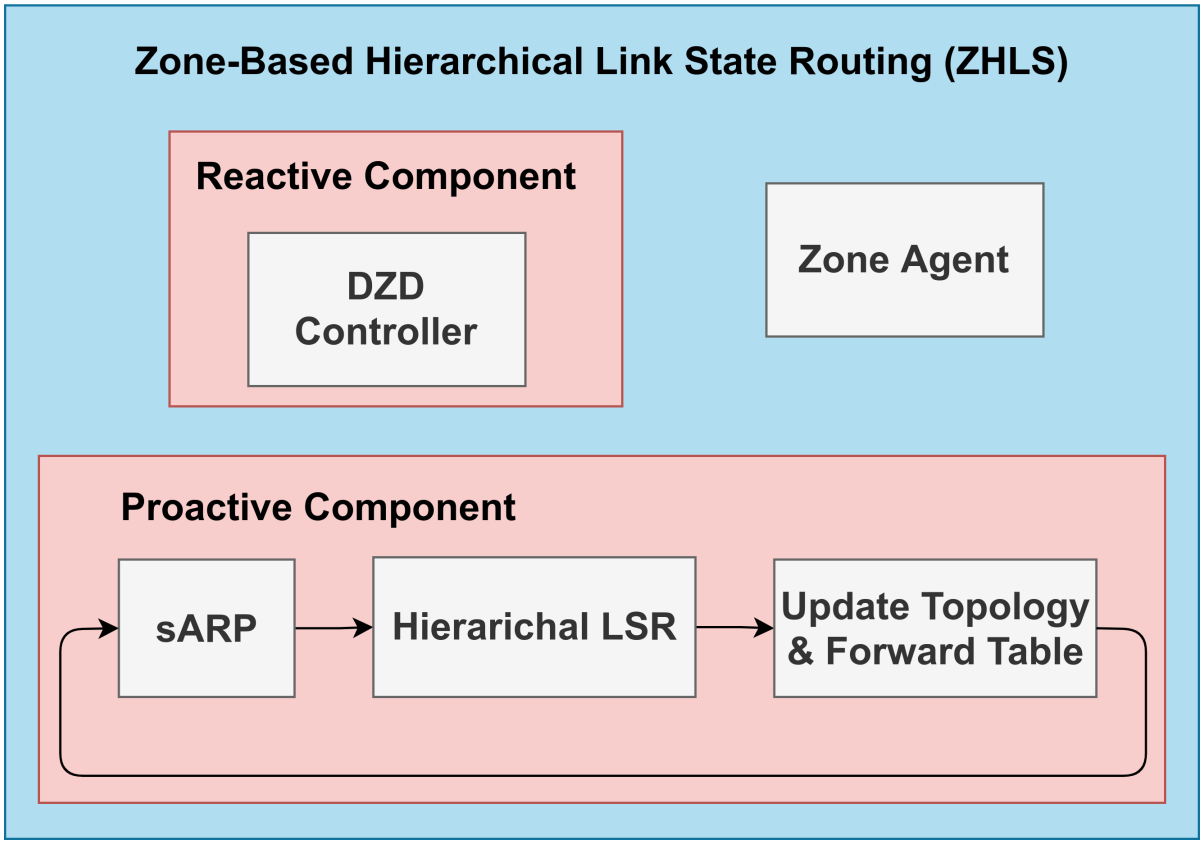

Figure 4.5: ZHLS Modules

Zone Hierarchical Link state Routing (ZHLS) may be divided into proactive and reactive components due to the fact that it is a hybrid routing protocol. In addition, there is a module that is in charge of zone administration. ZHLS modules can be show in Figure 4.5 The proactive component of the ZHLS is the component that is repeated on a regular basis to gather and store up-to-date routing information about the topology and state of zones. It does this by following three consecutive protocols: collecting data about neighbor nodes and zones which is done using Secure ARP protocol, flooding this information over the network using LSR packets, and updating the hierarchical topology and forward table with the flooded information. This procedure is done on a regular basis to maintain the routing information updated. The reactive component of the ZHLS is the component that is responsible for discovering the destination zone as soon as data is ready in the source and the destination IP doesn't exist in the source forwarding table (the destination is in a different zone). The protocol that is used for this task is called Destination Zone Discovery (DZD). Finally, the Zone Agent module is in charge of managing the zones' regions and locations. It splits the whole globe into a series of variable-sized zones and uses the position of each router to determine which zone it is in.

4.4.2.1 Zone Agent

In ZHLS, the map is divided into zones of a fixed size. In our implementation, we took it one step further and designed a variable-size zone scheme. Each zone is defined by its Cartesian coordinates, x and y. The size of each zone depends on the number of bits allocated to store the x and y coordinates, what we call the ZLength. The maximum ZLength is 16 bits, which gives the smallest zone size of 1.223 x 1.221 km^2. Different nodes may have different views of the world depending on their ZLengths as illustrated by Figure 4.6. For example, if node A uses a ZLength of 14, node B uses a ZLength of 15, and node C uses a ZLength of 14, then what node A perceives as one zone is perceived as 4 zones by node B and 16 zones by node C.

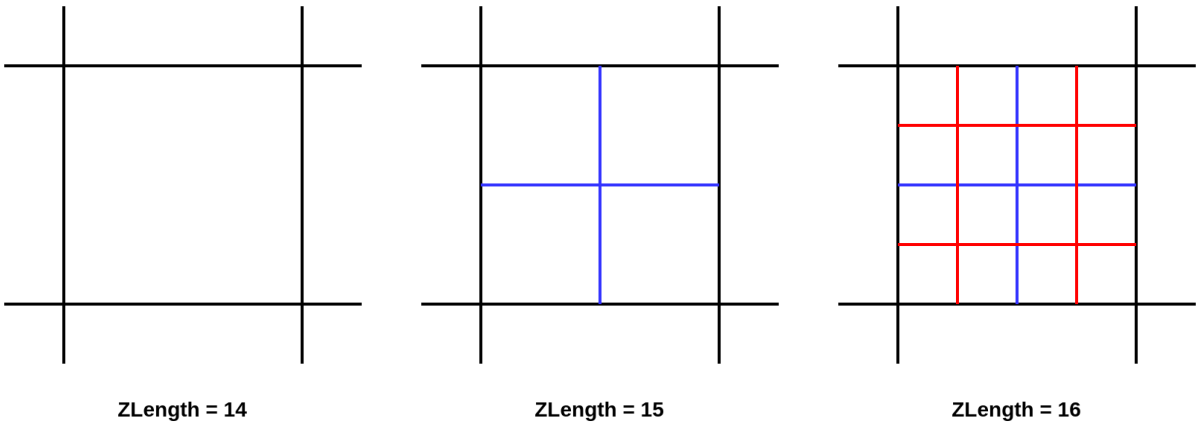

Figure 4.6: Zones as Perceived by Node of Different ZLengths

The zone agent connects to a GPS device using a Unix socket, and uses it to continuously read the node's latitude and longitude. The latitude and longitude are mapped from degrees to 16-bit integers to calculate the x and y coordinates. Afterward, the ZLength most significant bits of the coordinates are concatenated to form the Zone IDentification Protocol (ZID). If the zone agent detects that the zone is changed, it informs other submodules (subscribers) to take appropriate actions according to their logic.

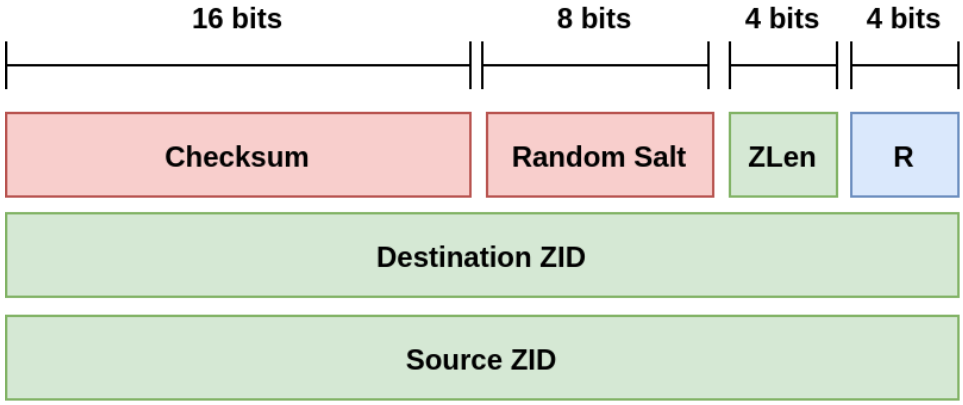

Figure 4.7: ZID Header

The zone agent keeps track of the current zone of the node for other submodules to use. It also provides some utilities to calculate distances between zones, check if two zones are

intersecting (as they may have different ZLengths), check if two zones are exactly the same, and calculate the area of a zone (which is used by Variable Radius Broadcasting). Since it holds the information of the node's current zone, other submodules use it to create **ZID Headers**. In our implementation of ZHLS, a ZID Header is prepended to most control and data packets. It contains information about the source and destination zones, as shown by Figure 4.7. The header fields are as follows:

- **ZLen:** The ZLength of the Source ZID and Destination ZID fields.
- **Destination ZID:** The Zone ID of the destination.
- **Source ZID:** The Zone ID of the source.
- **Random Salt:** Randomly added bits so that the encryption of a ZID header with the same source and destination nodes varies.
- **Checksum:** A checksum of the previous fields used to validate the ZID header.

4.4.2.2 Secure ARP

Address Resolution Protocol (ARP) is a protocol used in IP networks to map layer-3 IP addresses to layer-2 MAC addresses. ARP is vulnerable to MAC spoofing attacks, where someone in the network masquerades as someone else. Since C4IAN has many use cases where communications security is critical, we decided not to use the ARP protocol implementation in Linux. Instead, we implemented a secure version that we call Secure ARP.

Secure ARP is a simple protocol, similar to ARP, that is used in our system for three reasons:

- Identify the direct neighbor hosts (or neighbor zones) of any given host.
- Find the IP address to MAC address mapping for those direct neighbors.
- Calculate the latency for communicating with each neighbor node.

The main reason that Secure ARP can use encryption to secure its communications while ARP cannot, is that all nodes in a tactical team network can easily share a secret key. This is not the case in any IP network, where the users may have no idea who else is present on the same network.

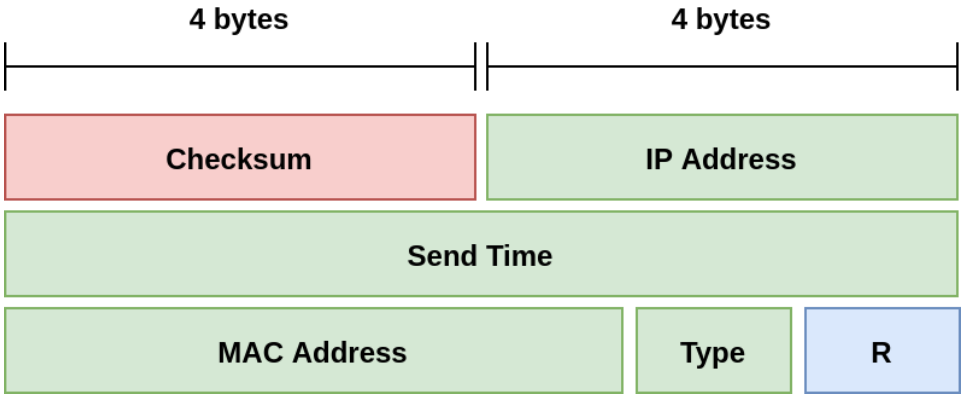

Figure 4.8: SARP Header

In Secure ARP, all packets consist of a ZID header and a SARP header that is shown by Figure 4.8. The header fields are as follows:

- **Type:** SARP packets are either a *SARP Request* or a *SARP Response*.
- **IP address:** The IP address of the sender.
- **MAC address:** The MAC address of the sender.
- **Send Time:** The time at which the packet was sent.
- **Checksum:** A checksum of the previous fields used to validate the SARP header.

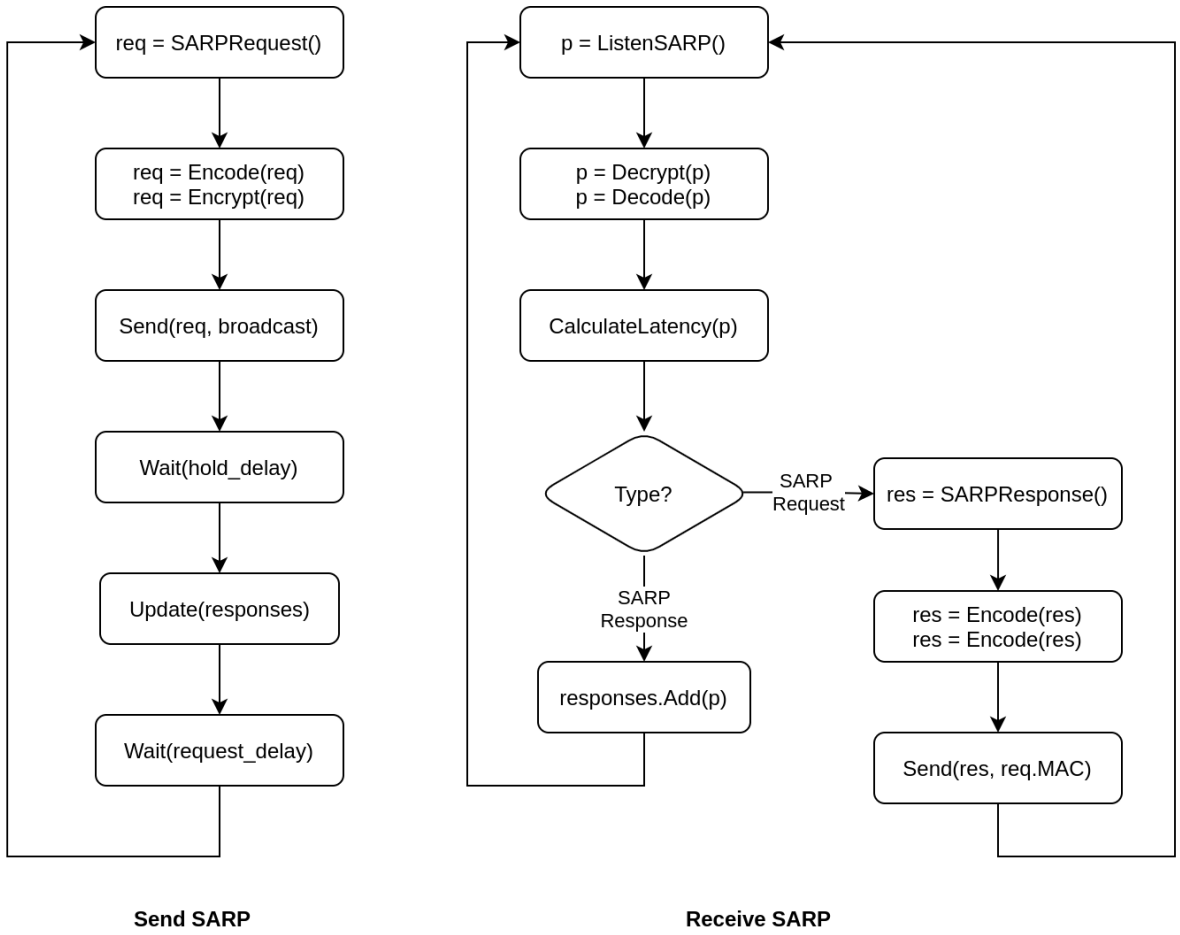

Figure 4.9: SARP Protocol

SARP operates through the two concurrent routines *Send SARP* and *Receive SARP* shown by Figure 4.9. Some details are omitted for conciseness. For example, when a SARP packet is received, it is actually processed in a different routine than the listening one to avoid missing a SARP packet while processing a previous one.

The SARP protocol operates by periodically sending and receiving SARP requests and responses to find the node's direct neighbors and their information. It maintains a *Neighbors Table* that contains the Node ID (IP or Zone ID), MAC address, and most up-to-date latency of each direct neighbor node or zone. When a neighbor belongs to a different zone, it is considered as a neighbor zone rather than a neighbor node. This is determined by the ZID header. The *Neighbors Table* is used by the Unicast Controller, as will be shown later in this chapter.

The *Send SARP* routine starts by creating an SARP Request, encoding it into binary, and encrypting it. Afterwards, it sends the packet through the MAC layer to all nodes within

range. The routine then waits for a hold period, giving a chance for neighbors to respond. The responses collected by the *Receive SARP routine* are used to update the Neighbors Table. The routine then waits again for an inter-request delay before sending the next request. The *Send SARP* routine also detects whether the neighbors have changed between subsequent requests by comparing hashes of the Node IDs in the Neighbors Table before and after it collects responses.

The *Receive SARP* routine continuously listens for SARP packets. When a packet is received, it is decrypted and decoded. The latency between the current node and the sender is then calculated. If the received packet is an SARP response, it is collected for the operation of the *Send SARP* routine. Otherwise, if is an SARP Request, a response is created, encoded, encrypted, and sent through the MAC layer to the sender.

One last missing detail is how *CalculateLatency* procedure calculates the latency information between the current node and the sender node using the send time in the SARP header. To calculate the latency accurately, both nodes must be in sync. Since most tactical devices have a GPS, we assume that the GPS is used for time synchronization.

4.4.2.3 Hierarchical LSR

Hierarchical Link State Routing (LSR) is the method through which nodes exchange information about the network. It is based on Link State Routing (LSR), in which every node constructs a connectivity graph of the whole network. However, in ZHLS every node only maintains information about the nodes inside its own zone, in addition to zone connectivity information. It consists of two main components: Intrazone LSR and Interzone LSR.

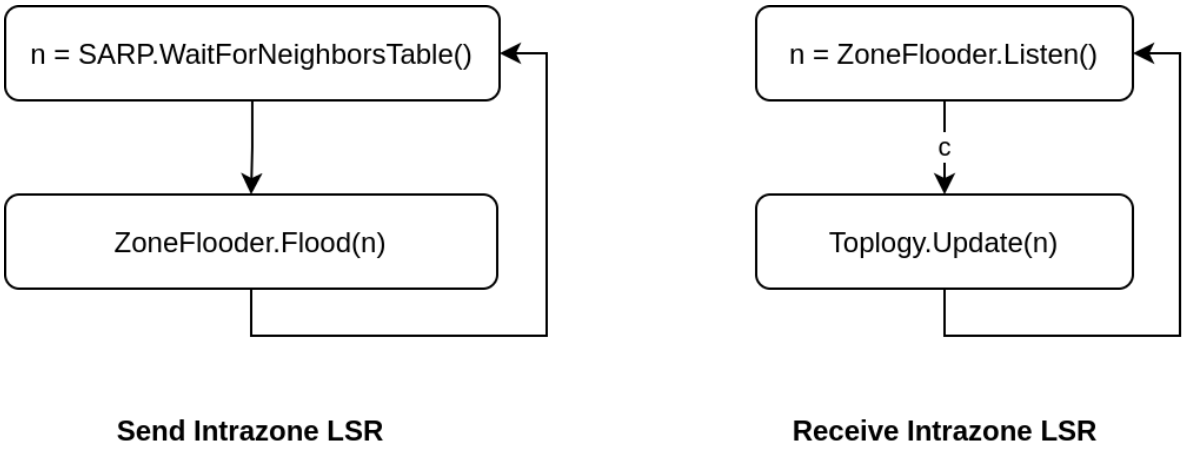

Figure 4.10: Intrazone LSR

In Intrazone LSR, a zone flooder is used to flood the Neighbors Table produced by Secure ARP every time it is updated. All nodes in a zone receive Intrazone LSR updates periodically from each other and compute connectivity information about all nodes in the zone, and the connectivity of their zone to its direct neighbor zones. Intrazone LSR is illustrated by the *Send Intrazone LSR* and *Receive Intrazone LSR* concurrent routines shown by Figure 4.10.

In Interzone LSR, a global flooder is used to flood a special Zone Neighbors Table to all the nodes in the network periodically. A Zone Neighbors Table of a zone contains the zones that it has direct connectivity to. All nodes in the network receive Interzone LSR updates from all zones and compute connectivity information about all zones. Interzone LSR is illustrated

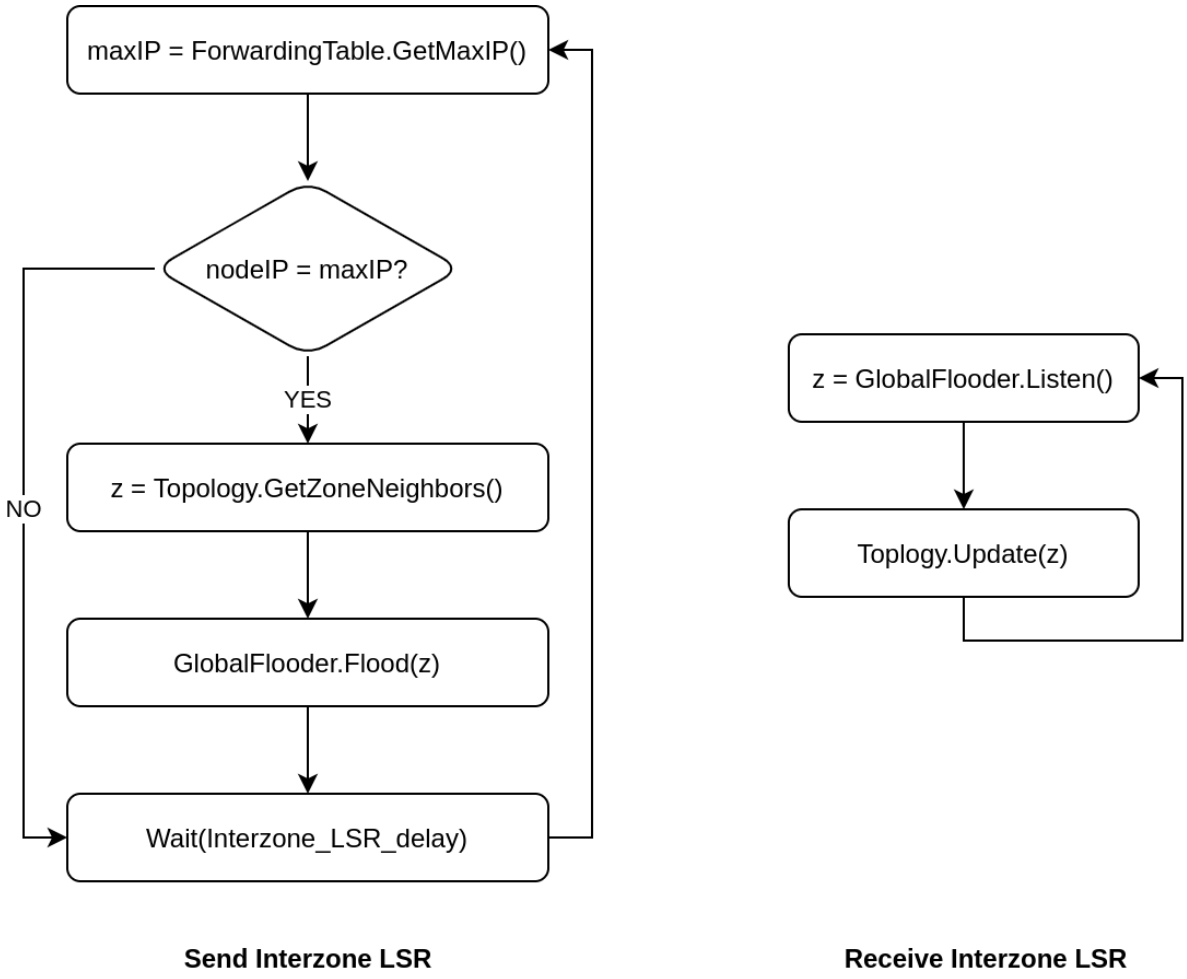

Figure 4.11: Interzone LSR

by the *Send Interzone LSR* and *Receive Interzone LSR* concurrent routines shown by Figure 4.11.

A question that naturally rises in Interzone LSR is which node is responsible for sending Interzone LSR in each zone. In the original ZHLS paper, all nodes flood Interzone LSR packets. However, this incurs a huge bandwidth overhead. In our implementation, we designed a simple scheme to avoid this. Since all nodes in a zone know the IP addresses of each other (through Secure ARP and Intrazone LSR), they can all agree on which node has the highest IP address. Hence, all nodes in a zone delegate the task of flooding Interzone LSR packets to the node with the highest IP address. One might argue that with mobility and packet loss, nodes may disagree on which node has the highest IP address, causing multiple ones to send Interzone LSR packets. Even though this case might occur only temporary, having two or three nodes flooding Interzone LSR packets is still better than having all the nodes in the zone do it.

4.4.2.4 Hierarchical Topology Graph

The topology graph is a major component of any proactive or hybrid routing protocol that is used to keep up-to-date routing information and is used to compute the best path from some source (usually the router itself) to any reachable destination. Zone Hierarchical

Link state Routing (ZHLS) is divided into two topologies: node-level and zone-level. Consider the case when you require a path to a distant location (Interzone routing). The source node must consider both node-level and zone-level topologies and use the shortest route method on both. Aside from the memory requirements for both topologies. So we reasoned that merging both topologies into a single one with distinct types of nodes would be more efficient.

Topology graph in C4IAN has two different type of vertices, nodes and zones as show in Figure 4.12. Node vertices (represented by circles) represent nodes that exist in the same zone with the router itself and each one is identified by the node IP, while zone vertices (represented by squares) represent zones that are reachable from that router and each one is identified by the zone ID.

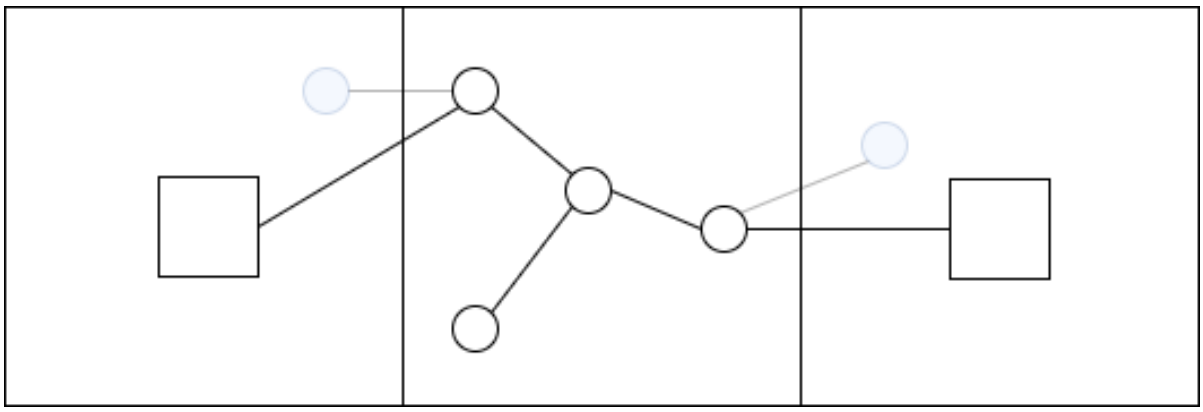

Figure 4.12: Hierarchical Topology Graph

The following two approaches are used to keep the topology graph up-to-date for all nodes and zones routing information:

- Periodically receives Intrazone and Interzone LSR packets as illustrated previously and uses this information to update the topology graph vertices, edges, and weights.
- Each vertex in the graph should have a timer attached to it (nodes and zones). Once we have received an LSR packet from that vertex, reset the timer (Intrazone LSR packets for node vertices and Interzone LSR packets for zone vertices). Remove this vertex from the topology graph instantly once the timer goes off.

Keeping the topology graph up-to-date is mandatory as it will be used mainly for two important functionalities:

- Find the shortest paths by applying the Dijkstra algorithm (using latency metric) to all nodes and zones in the graph and use this information to fill the forwarding table.
- Find all reachable neighbor zones by applying controlled depth-first search from the router vertex till reaching zone vertices and use this information to apply zone-level broadcasting, which is much more efficient than normal flooding.

4.4.2.5 Destination Zone Discovery

Destination Zone Discovery (DZD) is a major module in Zone Hierarchical Link state Routing (ZHLS). Consider the case when the source node require a path to a node that is outside its

zone (Interzone routing). The first step is to discover the zone of the required destination.

Destination Zone Discovery (DZD) protocol has two types of control messages that control the searching process: DZD Request which is sent using zone-level broadcasting by the router that want to discover the destination zone and DZD Response which is sent using normal unicasting from the node that know the destination zone to the source node which its zone was already stored in the DZD Request header.

DZD Request control message must contain two headers: ZID header which has the following fields:

- **Src Zone:** The zone ID of the source node which will be used later to send the DZD Response to.
- **Dst Zone:** The zone ID of the neighbor zone to the source zone which is the DZD Request will be sent to.

And DZD Request Header which has the following fields as show in Figure 4.13:

- **Src IP:** The IP of the source node which will be used later to send the DZD Response to.
- **Required Dst IP:** The IP of the destination node which it's required to know its zone.
- **Visited Zones:** Array of all zones that the DZD Request message has already gone through. This array is used to prevent recirculation of the same message indefinitely.

Figure 4.13: DZD Request Header

The reachable neighbor zones are found using depth-first search on the hierarchical topology graph, and a copy of the DZD Request message is sent to each of them once the DZD Request message is generated using the preceding headers. Each zone will repeat the process until the destination zone is discovered.

The first node that receive the DZD Request message at the destination zone will stop broadcasting DZD Request message and start to form DZDResponse message using ZID header which contains:

- **Dst Zone:** The zone ID of the destination zone which, in this case, the zone of the source node that started broadcasting DZD Response message.

And DZD Response Header which has the following fields as show in Figure 4.14:

- **Dst IP:** The IP of the destination node which, in this case, the source node that started broadcasting DZD Response message.
- **Required Dst IP:** The IP of the destination node which it's required to know its zone.
- **Required Dst Zone ID:** The destination zone ID which the source node was looking for it.

Using the Dst IP in DZD Response Header and Dst Zone in ZID Header, DZD Respone packet can be unicasted to the original source that sent DZD Request message. Each node along the path of DZD Respone and the original source itself cache the destination zone for upcoming messages.

Figure 4.14: DZD Response Header

The original source can transmit all data buffered to that destination after receiving the DZD Response message. If the search fails, it will be repeated three times, each time after a certain amount of time, until the target zone is found, otherwise it will be announced as a failure and all stored packets for that destination will be discarded.

4.5 Multicast Controller

4.5.1 Functional Description

The multicast controller is in charge of many-to-many transmission between network nodes; that is, multiple senders and multiple receivers, each with an IP address. At every node, it creates a Multi-Forwarding table, which includes the best next-hops to forward a packet to for each possible multicast group. Collectively, this constitutes by periodically sending control packets join query and join replies and filling tables in every. As indicated in the preceding section, we believe On-Demand Multicast Routing Protocol (ODMRP) is one of the best multicast protocols for this task. As a result, we chose it to be our multicast routing protocol.

4.5.2 Modular Decomposition

On-Demand Multicast Routing Protocol (ODMRP) may be divided into two part, the control packet generator and the tables. In addition, there is a flooder module is in charge of flooding the join query packets and timers queue for deleting expired table entries from the tables. ODMRP modules can be show in Figure 4.15

4.5.2.1 Control Packets Generator

Generating right control packets is critical for this protocol as ODMRP depends on the control packets for filling the tables, construct the routes and the network soft state maintenance. This module generates two important control packets called Join Query (format illustrated in Figure 4.16) and Join Reply (format illustrated in Figure 4.17).
A Join Query is generated when a multicast source has packets to transmit but no route and group membership is known, a Join Query packet contains the following fields:

- **Dests Count:** The Number of destinations IPs.
- **Checksum:** It is used to validate the received control packet.
- **SeqNo:** Unique sequence number of the control packet.
- **TTL:** Time to live, which is the number of hops this packet can traverse.
- **SrcIP:** Source IP address where the control packet is generated.
- **GrpIP:** Group IP address of this multicast group.

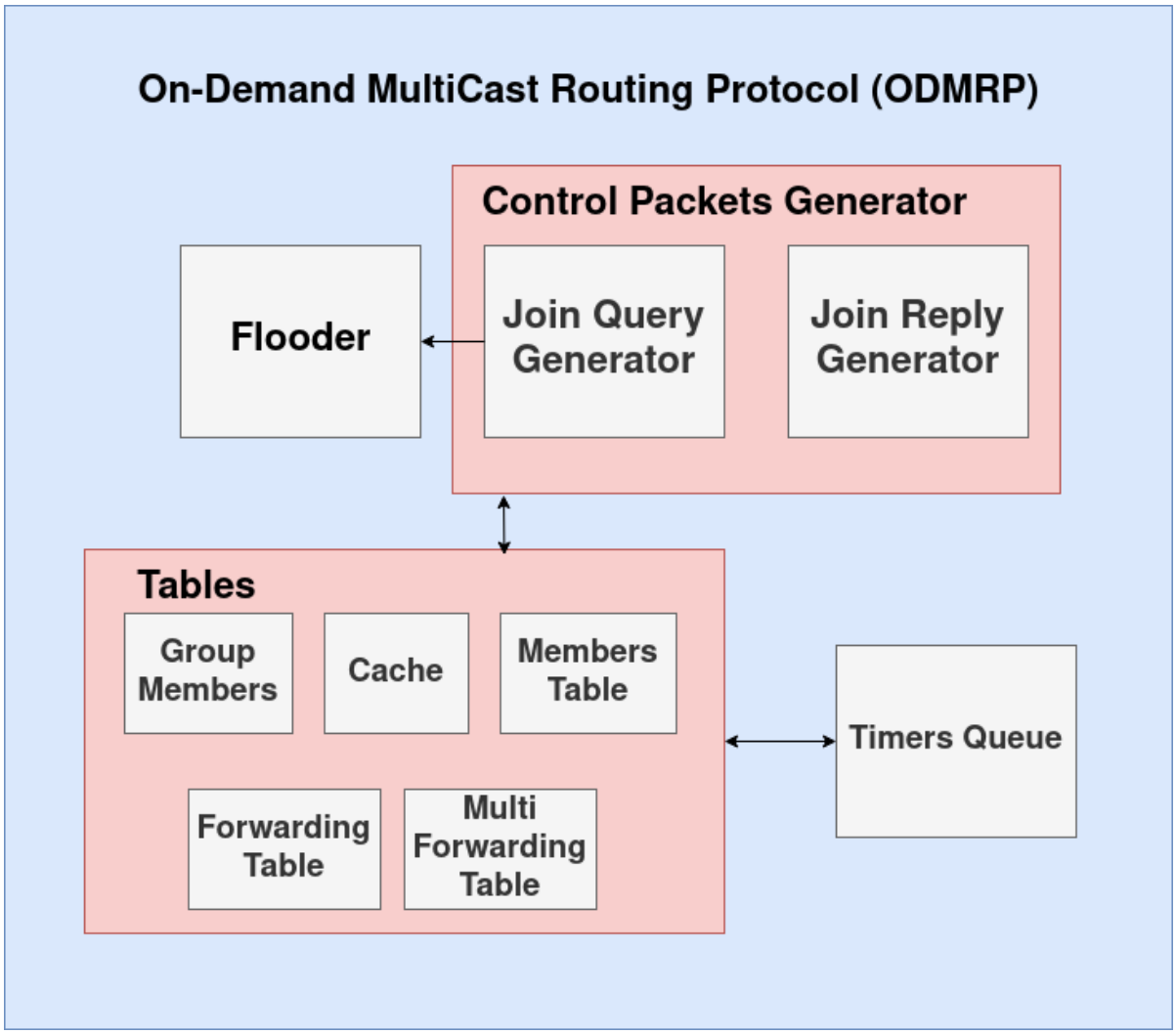

Figure 4.15: ODMRP Modules

- **PrevHop:** Mac address of the last node which has processed this packet.
- **DestIPs:** The destination IPs of this multicast group.

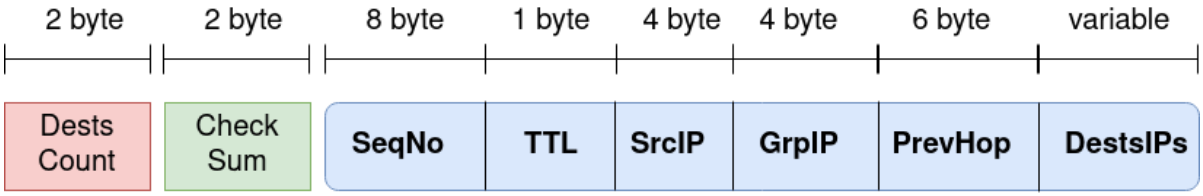

Figure 4.16: Join Query Packet

Join Query control packet is generated using the following:

- **SeqNo:** Incremental value in the multicast control module, different for each control packet sent.
- **TTL:** Default time to live value in our case we set it to 100.
- **SrcIP:** This Node IP where the join query packet is generated.
- **GrpIP:** Group IP address of this multicast group.
- **PrevHop:** MAC of current node.
- **DestIPs:** The destination IPs of this multicast group IP.

Then this Join Query packet is flooded using the flooder and gets updated and update the node's tables until it reaches a destination the destination node generates a Join Reply control packet which contains the following fields:

- **Next Hops Count:** Next Hops MAC addresses count which equals the source IPs address count either.
- **Checksum:** It is used to validate the received control packet.
- **SeqNo:** Unique sequence number of the control packet.
- **DestIP:** Destination IP where the Join Reply is generated.
- **GrpIP:** Group IP address of this multicast group.
- **PrevHop:** Mac address of the last node which has processed this packet.
- **SrcIPs:** Source IPs addresses where the control packet is generated.
- **Next Hops:** Next Hops MAC addresses to reach the sources.
- **Cost:** Cost of this path, which equals the number of hops.

Join Reply control packet is **generated** using the following:

- **SeqNo:** Received join query sequence number.
- **DestIP:** IP of the current node where the Join Reply is generated.
- **GrpIP:** Group IP address of this multicast group.
- **PrevHop:** Mac address of the current node.

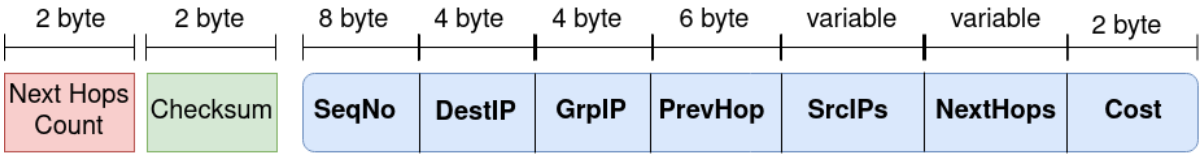

Figure 4.17: Join Reply Packet

- **SrcIPs:** Filled using cache table by iterating on cache table entries and checks if the group IP is equal to the group IP of this multicast group, if true it appends the source IP found in the cache table to this join reply control packet.
- **Next Hops:** Filled using cache table by iterating on cache table entries and checks if the group IP is equal to the group IP of this multicast group, if true it appends the next hop found in the cache table to this join reply control packet.
- **Cost:** 1.

Then Join Reply packet is sent to the next hops addresses and gets updated and update the tables then passed to the next hops addresses until it reaches the multicast sources.

4.5.2.2 Tables

As we knew in Chapter 3 ODMRP are mesh based and soft state protocol, so tables play big role in ODMRP protocol to construct the mesh, forwarding group members and the routes to reach the destinations nodes.

Group Members Table is the table which is used to hold the multicast group IP address and the corresponding destinations IP addresses So a packet can be sent using only the group IP, Figure 4.18 represents one entry of Group Members Table, It is filled by a configuration file in the senders nodes.

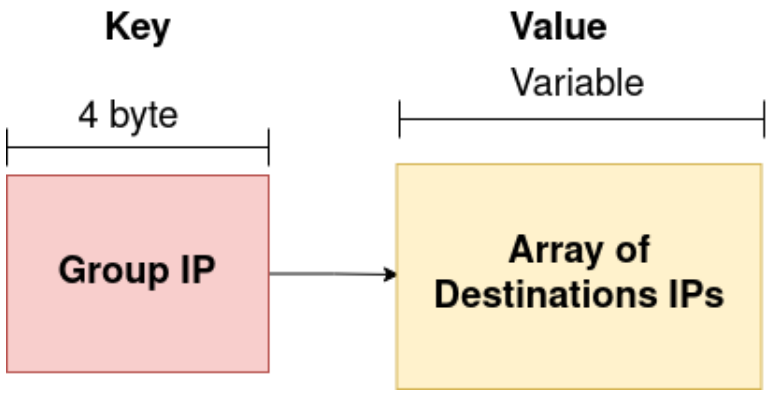

Figure 4.18: Group Members Table

Cache Table is the table that holds the control packet source IP and the corresponding (Sequence Number, Group IP, Previous Hop and cost). And the table is used to:

1. Check for control packet duplicates.
2. Hold cost value to check if a better entry has been found.

3. Generate and update join reply control packet.

Figure 4.19 represents one entry of Cache Table.

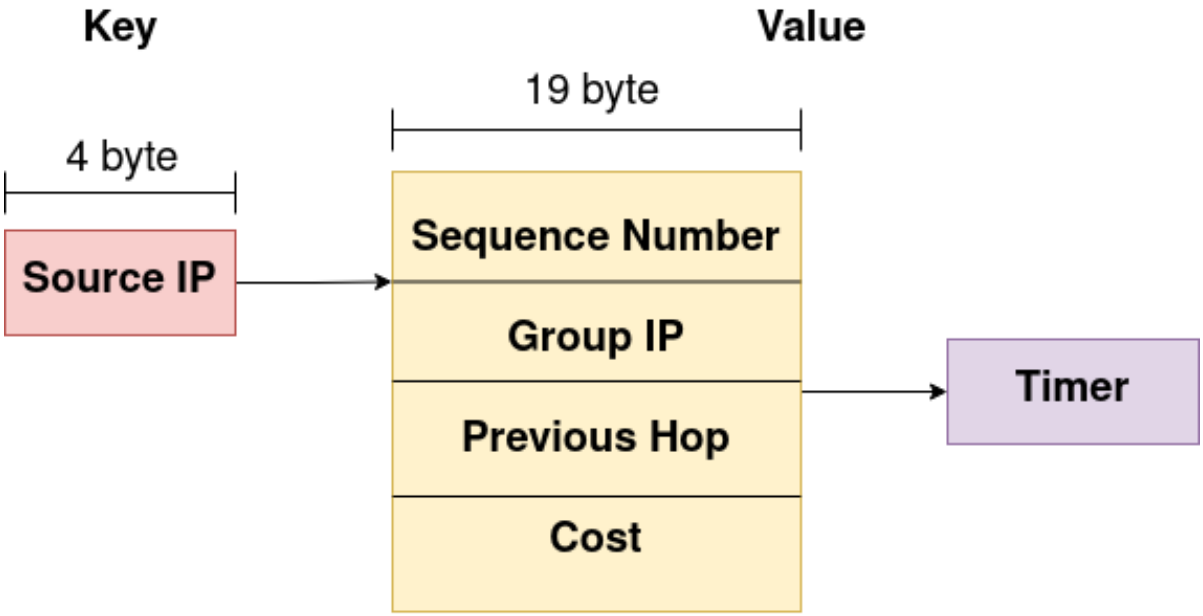

Figure 4.19: Cache Table Entry

Members Table is a table that holds the group IPs that the current node is a part of. This table for faster checking that the current node is a destination of a certain multicast group IP address, and it gets filled the first time the node knew it is a destination of a certain multicast group IP address. Figure 4.20 represents one entry of Members Table.

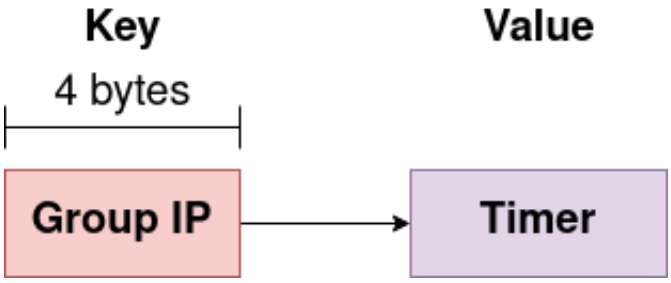

Figure 4.20: Members Table Entry

Forwarding Table is a table that holds some entries, each one has a destination IP address and the corresponding next hop MAC address and cost. It gets filled when receiving a join reply from a destination, and it is used to update the Multi Forward Table (more details in the next section 4.5.3). Figure 4.21 represents one entry of Forwarding Table.

Multi Forward Table is a table that holds some entries, each one has a multicast group IP address and the set of corresponding next hops to reach this multicast group IP destinations. It gets filled when receiving a join reply from a destination (more details in the next section 4.5.3), and this table is used by the node router to direct the packets to reach

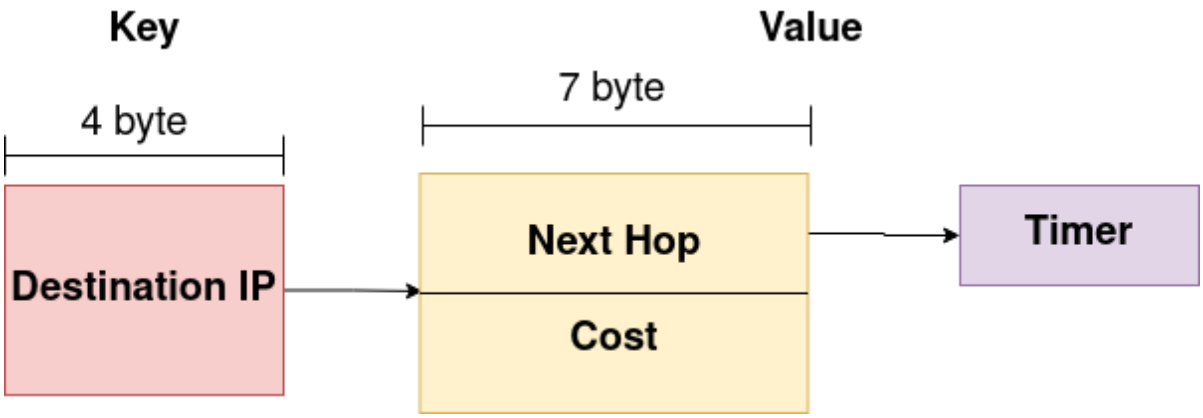

Figure 4.21: Forwarding Table Entry

destinations node of a certain multicast group IP address. Figure 4.22 represents one entry of Multi Forward Table.

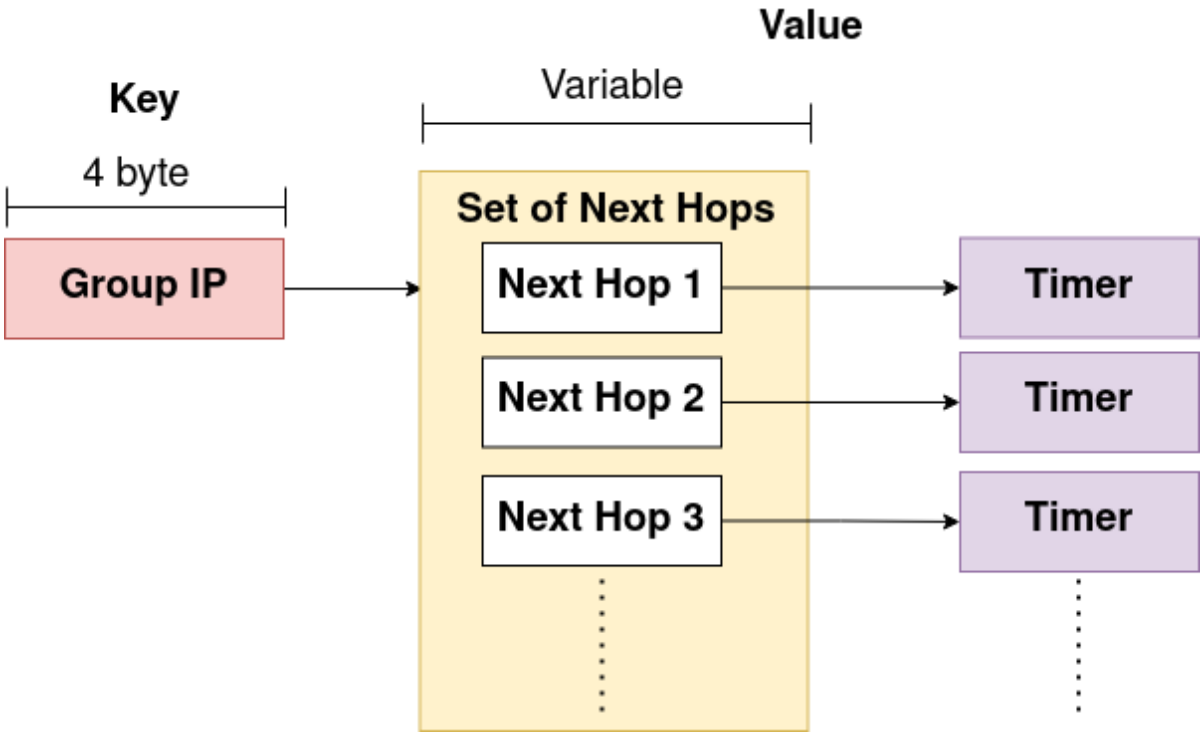

Figure 4.22: Multi Forward Table Entry

4.5.3 ODMRP Protocol Flow

In ODMRP [18], group membership and multicast routes are formed and updated by the source on demand. When a multicast source has packets to transmit but no route and group membership is known (Multi Forward 4.22 Table is empty), it floods an advertising control packet. This packet, named Join Query (format illustrated in Figure 4.16) and it is regularly sent by the flooder to the whole network to refresh the membership information and update the routes.

When a node gets a Join Query packet, time to live is decreased by one, and it records the source address and the unique identifier of this control packet to its Cache Table 4.19 to identify duplication to construct the join reply later. If the Join Query control packet is not a duplicate by verifying it is not a duplicate from the cache table and the Time to live is higher than zero, the cache table is updated, and it is rebroadcast.

When a Join Query packet reaches the multicast receiver, it adds the group IP to its member table as an entry (Figure 4.20) so the node realizes that it is a part of this forwarding group with a specific group IP address.
The destination node then generates a Join Reply control packet (format shown in Figure 4.17) with the help of the cache table to add next hops and source IPs which are filled before by the received join queries and sets the cost of the generated join reply to be one.

Then sends the generated join reply to the previous hops nodes, The classical ODMRP protocol floods the join replies packet until it reaches all the sources, We did an **optimization** by sending the join reply via the next hops stored in the cache and making the join reply propagated and received by only the forwarding group members and not broadcasted to any other node which are not a part of this multicast group.
After generating the join reply control packet sends the join reply with the same manner by updating the source IPs, next hops, the cost, update the forwarding table 4.21 and also construct the routes by filling the multi forward table 4.22 with the next hops MAC addresses corresponding to this group IP address. Updating the route in the Multi Forward Table is done by checking if the forwarding table is updated by a newly added destination IP or destination IP address exists but the new received join reply cost is less than the existing one. If this is true, both the forwarding table and multi forward table are updated by adding the previous hop of the received join reply to the Next hops set, using the group IP as the key of the next hops.

This method generates a mesh of nodes with forwarding group members and routes from sources to receivers until it reaches the multicast source through chosen paths.
When the node's router wants to send a multicast packet is checks if Multi Forward Table has entries using this multicast group IP address as the key, and it can access a set of next hops to send the packet to them, so it can do multicasting.

4.6 Command Center Client

Command centers are responsible for the central administration and operational management of a group of units. High-end computers with strong CPUs, high power consumption capabilities, and large storage and RAM capabilities are expected to be used as command center clients. The command centers are located near the operation field and have a wide wireless

range, allowing them to link to a group of troops in the field. They're assumed to have limited (or no) mobility.

4.6.1 Functional Description

Command centers in C4IAN have the ability to collect data from a collection of units, show it to the operators or unis leaders, and control these units by delivering a series of instructions. What exactly a command center can perform is summarized briefly in the following list:

- Send audio command and command codes to a single unit.
- Send audio commands to a group (multicast) or everyone (unlimited-radius broadcast).
- Store all sent and received data.
- Show old data (audio, messages, videos, sensor data)
- Show notifications when an audio message or command code is received.
- Show video streams as they are received from units.
- Show a map of connected units distinguished by group color to each unit belonging to some group.
- If sensor data isn't received in 2 minutes, mark the unit as inactive and notify the command center operators.
- If a unit's heartbeat is below a threshold, mark it as in danger and notify the command center operators.

4.6.2 Modular Decomposition

As illustrated in Figure 4.23, C4IAN's command center is organized into two major modules: Daemon, which is in charge of sending and receiving data from other units in the network, and User Interface, which allows operation leaders to manage and visualize this data as well as record command audios and choose command codes that they want to send to these units.

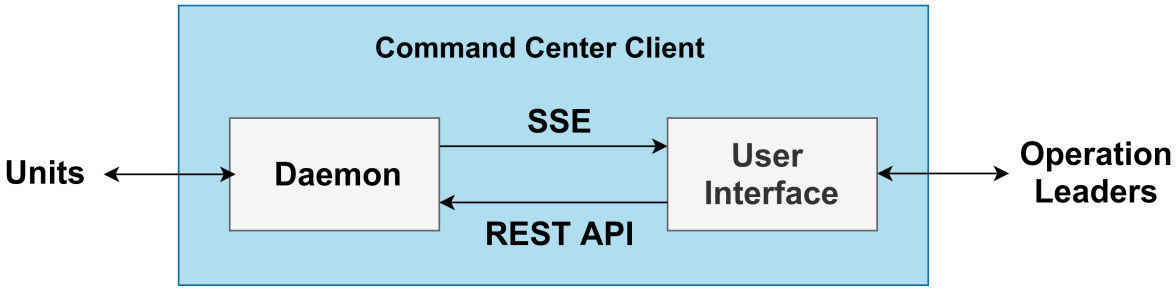

Figure 4.23: Command Center Client

Video Streamer is a submodule of both the Daemon and User Interface that will be discussed in better detail later.

4.6.2.1 Daemon

The daemon of the command center client is a highly concurrent program that runs in the background on the command center's devices. It exchanges code messages, audio messages with units, and receive sensors data and video streaming from the units through a network interface, runs a database to keep track of all sent and received data, and connects to the user interface through a RESTful API server.

On the network interface, the unit has two concurrent routines that listen for TCP connections and UDP datagrams. When receiving a TCP connection or a UDP datagram, the incoming data is handled in a separate concurrent routine to avoid missing connections or incoming datagrams while processing previous ones. Any data received is prepended with a byte that identifies the packet type. The packet payload is then decoded, the data is stored in the database, and the user interface is notified through Server-Side Events (SSE). The network interface also provides routines to send or receive packets over TCP or UDP to units, in case the command center needs to send code messages or audio messages to a unit, a group of units, or all the units. Note that all TCP and UDP data transfers are routed using our router.

The RESTful API server runs in a concurrent routine and provides endpoints for the user interface to collect and present stored data. In addition, it provides endpoints for the user interface to send code messages or audio messages to unit(s). All sent and received data are stored in the database. Since the database is accessed through different concurrent subroutines, mutual exclusion techniques are used to synchronize access to the database.

The main routine reads any necessary configuration or command line arguments on startup, initializes the database, then starts and orchestrates the aforementioned subroutines. Figure 4.24 shows a complete picture of the command center client daemon.

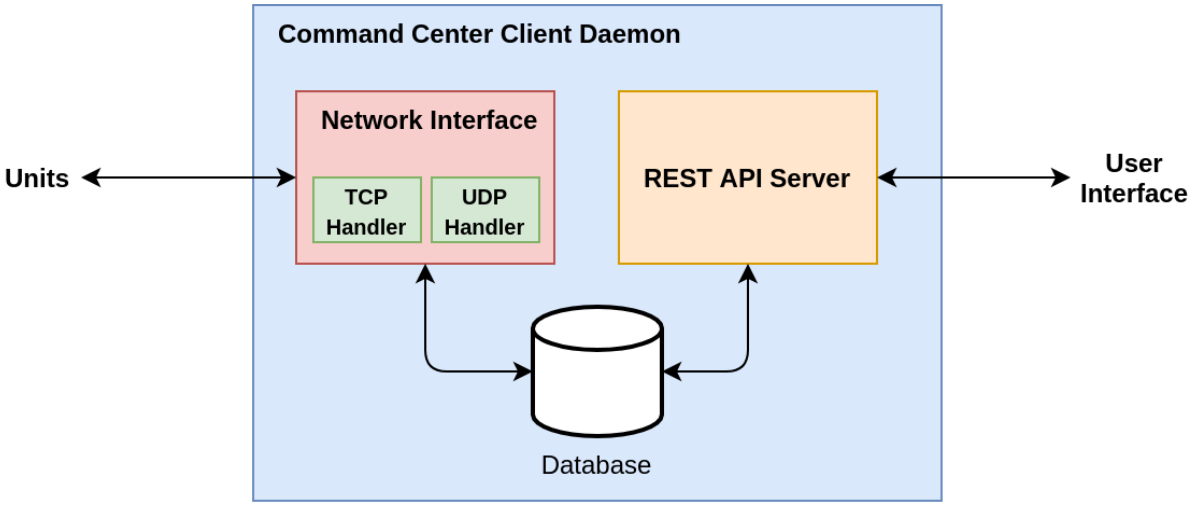

Figure 4.24: Command Center Client Daemon

4.6.2.2 Video Streamer

For video streaming, we use HTTP Live Streaming (HLS) protocol. While video streaming is running at a unit, the command center client receives a metadata file and video segment files from the units continuously. The video streamer receives those files through the daemon,

continuously updates the metadata, and stores the video segment files. In a dedicated media directory, the video streamer creates a directory per video stream per unit to store the video stream's metadata and video segment files and serve them to the user interface through a file server API which maps endpoints paths to local file system paths.

When a video stream starts, the video streamer notifies the user interface through a SSE. The user interface reads the metadata to identify the available video segments, and starts fetching the video segments and displaying the video stream continuously through the file server API. Figure 4.25 visualizes the data flow through the video streamer.

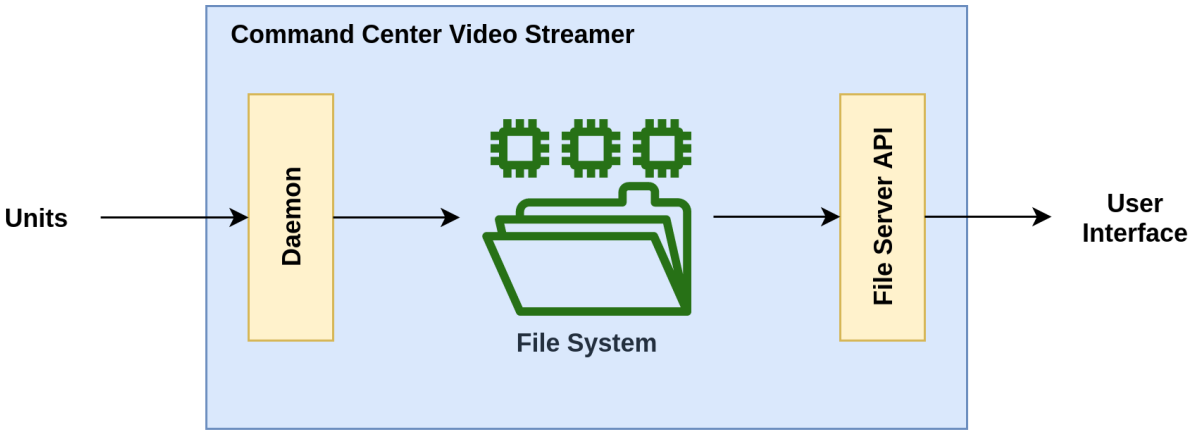

Figure 4.25: Video Streamer Data Flow

4.6.2.3 User Interface

The user interface in command center client allows operation leaders to manage and visualize units data, as well as record command audios and choose command codes that they want to send to these units. There is two-way communication between the command center client daemon and the user interface. Data that recently arrived at the command center daemon is sent directly to the user interface using Server-Side Events (SSE) to be visualized immediately to the operation leaders. While the data history is sent from command center daemon to user interface using Representational State Transfer (REST) API endpoints.

Command center user interface is divided mainly into four pages: a login page, a map page, a units page, and finally a streams page. The login page is used to authenticate the operation leader to prevent masquerade attacks, as shown by Figure 4.26.

The map page is where the operation leader can view all nearby units' precise positions on a real map, as shown by Figure 4.27. Each unit is color-coded according to the group it belongs to, assisting the operation leader in analyzing group movements. When some unit sensor data indicates that this unit is in danger, or when some unit was inactive for more than two minutes and did not provide any sensor data during that time, the operation lead is informed. Any communications received from neighboring units are shown in a chat window. As soon as the data reaches the user interface, there will live update at the command center's screen.

The operation leader can use the unit page (in Figure 4.28) to examine all of a unit's data history or send voice or message orders to it. All audios, videos, messages, position history, and sensors data of a unit are visible to the operation leader.

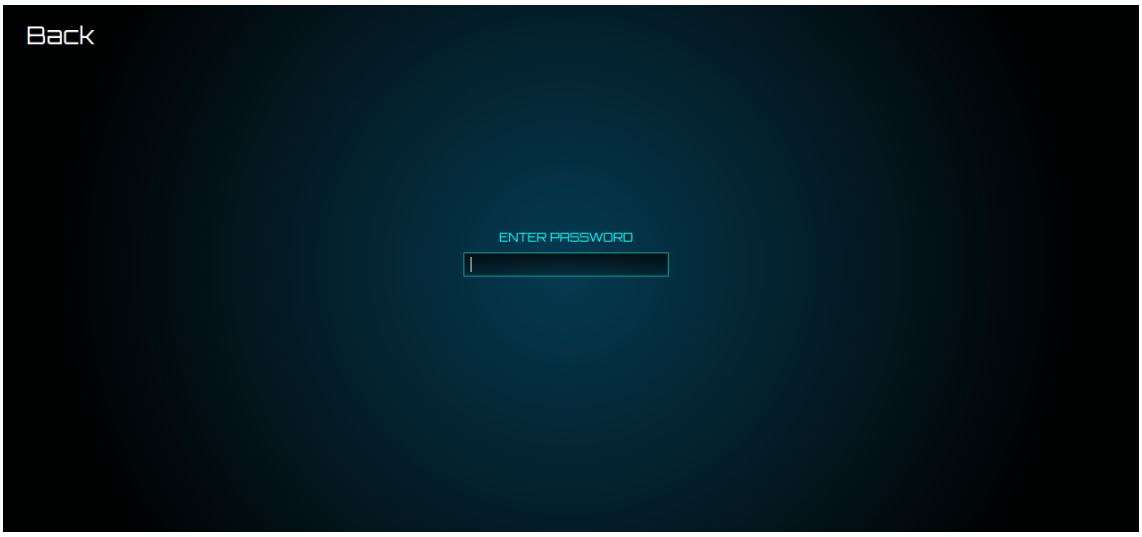

Figure 4.26: CMD UI Login

Figure 4.27: CMD UI Map

Finally, the operation leader may examine all live streams received from any nearby unit on the Streams page. This is shown by Figure 4.29

4.7 Unit Client

The unit client is the software that runs on the unit device, which is carried by the deployed soldiers and officers on the operation field. More detail about the unit device in 4.2.1.1.

4.7.1 Functional Description

- Stream video from combat cameras to command center(s) only if the latter requested them. Video streaming terminates if the unit received an end-stream request, or the

Figure 4.28: CMD UI Units

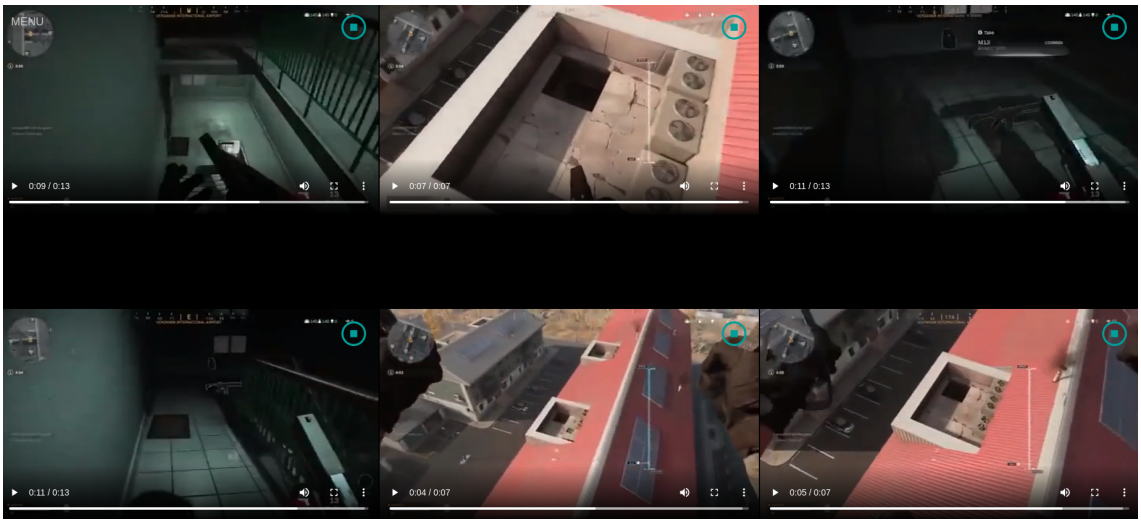

Figure 4.29: CMD UI Streams

start request wasn't refreshed after 1 minute.

- Stream the heartbeat and location of the device owner and their position every 10 seconds.
- Store all the recorded video and sensors (location and heartbeat) data locally.
- If the device user requested:
  - Send audio messages from the microphone.
  - Send code messages (every code has its predefined meaning.)
- Receive audio messages from command centers into a queue.
- Play received audio messages from the queue instantly.
- Receive and show code messages.

4.7.2 Modular Decomposition

Figure 4.30 shows the modules of the unit client.

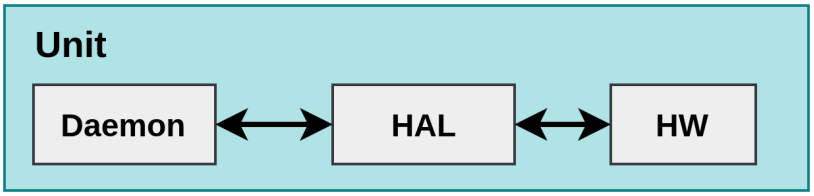

Figure 4.30: Unit Modules

The unit daemon implements the main state machine and logic of the unit. It communicates with the daemon of the command center and transfers messages, whether they were audio or codes or streaming requests.

Hardware Abstraction Layer (HAL), this abstracts the hardware, so we don't need to touch the unit client when porting to a different HW. It also serves the purpose of emulating the unit client HW without the unit being aware of that.

HAL is responsible for streaming video from the camera source, or a video file in case we are testing.

4.7.3 Design Constraints

The unit runs on battery powered devices, so it needs minimal power and CPU consumption.

We lowered the CPU footprint by requiring *FFmpeg* to generate only one level of quality for the video stream. Also, we require that the command center should send stream request periodically to keep the stream. This is useful in case the command didn't want a stream, but the unit didn't receive its end video streaming request.

Chapter 5

System Testing and Verification

This chapter introduces you to the testing setup and how we tested each module in isolation and the end-to-end integration that we planned and followed to prove to a high certainty the correctness of the system.

Testing ad-hoc networks always has been very tricky subjects, with researchers spending big amounts of funds on building testbeds with actual hardware and workers who move the hardware/devices around to change the connectivity between them. It is a tedious task to test ad-hoc protocols on actual hardware. The problem is harder when you want to execute the test multiple times. It is very costly in terms of money and time and other resources.

This is why we took the decision, which some researchers take, and aimed to emulate the environment the system would likely be deployed on. Most researchers take the path of network simulation, the difference is that with simulation, results are less accurate because of time compression and the lack of real world constrains like power consumption and latency while communicating with the kernel. With emulation, we can run the protocol implementation in real time while integrating it with real kernel. This enables us to explore how routing solutions could be built and constructed for real world systems, and let us face real world challenges regarding routing implementation. Also, with emulation, we can run real applications on top of the router to test its performance.

Nevertheless, emulation lacks the capability of spawning big number of nodes due to the limits of computer's memory and because each node is an entire VM/container with real programs. Kernel scheduler and memory management put limitations on the number of nodes' wireless devices we can emulate.

The next sections will walk you through our setup, scripts, GUIs and plans to prove the correctness of the programs we built in C4IAN.

5.1 Testing Setup

5.1.1 Mininet-Wifi

Mininet-Wifi is a python program and library that emulates wireless networks in infrastructure and ad-hoc modes. It's a wrapper around Mininet project, which aims to emulate software defined networks for research and education. It relies on *wmediumd* and *mac80211_hwsim* kernel modules to simulate wireless radio devices and uses network namespaces feature from Linux to isolate processes as if they are in their own device.

With *wmediumd*, Mininet-Wifi can simulate the propagation of wireless signals and control the probabilities of dropping packets and adding noise to them.

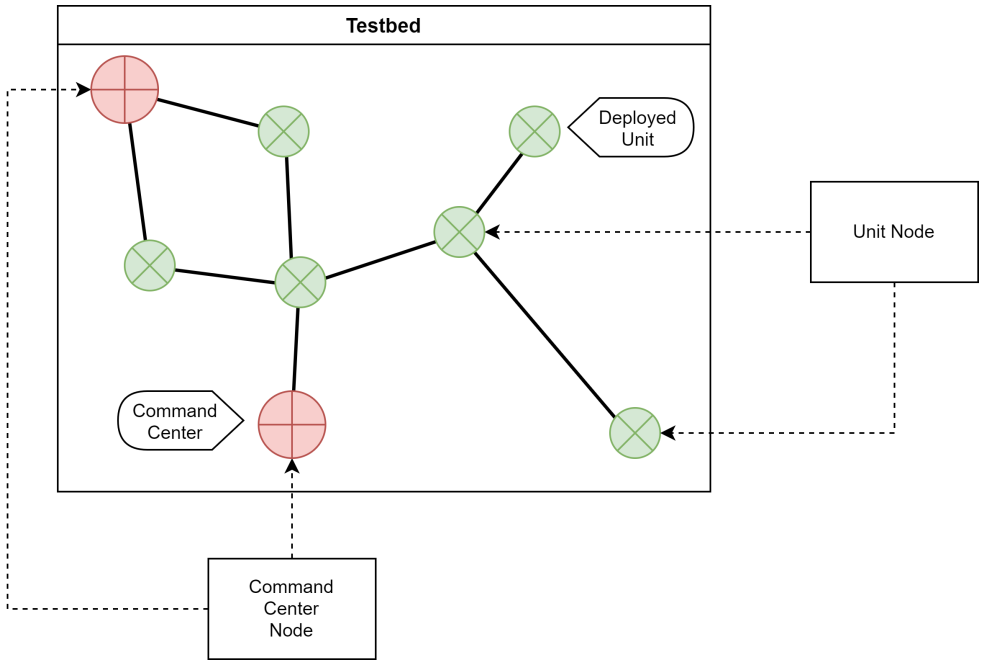

Figure 5.1: Mininet Testbed With Different Nodes

Figure 5.1 shows unit and command nodes running inside a testbed created by Mininet, each node should have a router process.

5.1.1.1 Installing Mininet-Wifi

Run the following:

```
$ python --version
```

If it's 3, you are good to go, otherwise do the following:

```
$ sudo mv /bin/python /bin/python.old
$ sudo ln /bin/python3 /bin/python
```

It may break your system, in this case reverse it back:

```
sudo m /bin/python.old /bin/python
```

Then install mininet-wifi:

```
$ (
    set -e
    sudo apt update
    sudo apt install -y git
    cd /tmp
    git clone git://github.com/intrig-unicamp/mininet-wifi
    sudo mininet-wifi/util/install.sh -Wln
)
```

5.1.2 Edit GUI and MN Script

To be able to create, edit and visualize the topology either offline or online (while Mininet-Wifi is running), we created *edit* which is a web GUI for the testbed.

Figure 5.2 shows edit UI, it has a map with units each has its name and range, you can move/add/delete nodes and change the hierarchical zones *Zlen* and apply a mobility simulation with some configuration while connected to Mininet.

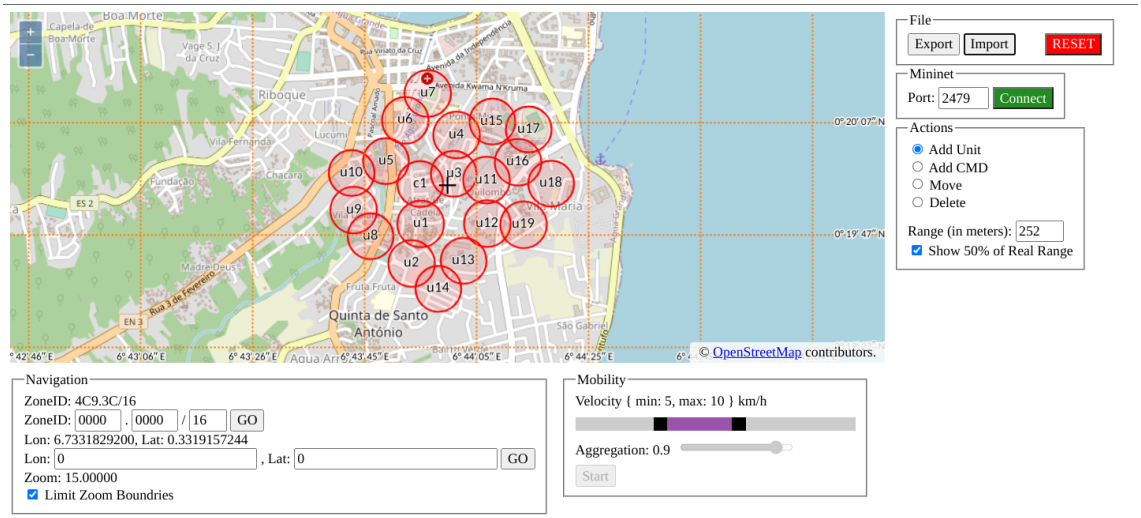

Figure 5.2: Edit GUI

Edit can export and import the topology into a specific format we made for the testbed. Also edit can connect to a script we wrote that controls the routers and units and Mininet, edit gets updates about nodes' locations in case the script is moving them (while in mobility mode) or edit sends location updates in case the user is moving the nodes manually.

The script - called mn, written in python - uses a modified copy of pymobility which implements "Time Variant Community" Mobility or TVC for short, which works by defining a group of units and the model keeps them around some moving center it calculates. This mobility model resembles the soldiers' units moving around, because most probably they won't walk randomly in different directions, but rather in groups together.

With "aggregation" control switch, you can in edit specify how much the group should move together, where 0 aggregation means every node is on its own.

The mn script sends the locations (longitude and latitude) for each node periodically through Unix sockets open by routers and units on the nodes.

5.1.3 Start Script

Start is a script that can any component of the system, whether it's on the localhost or inside the Mininet namespaces. The following is how to run Mininet on some topology and routers and units and command centers' daemons and HALs using start:

```
## (1st terminal)
# start mininet-wifi within chromosome topology
$ sudo ./start mn topos/chromosome.topo

## (2nd terminal)
```

```
# start routers in all nodes
$ sudo ./start routers

## (3rd terminal)
$ sudo ./start units
## (4th terminal)
$ sudo ./start cmds
## (5th terminal)
$ sudo ./start hals
```

Start lists Mininet namespaces using pgrep, then injects programs into the Mininet containers using nsenter and watches for file changes using watchdog script.

5.1.3.1 Socat

The CMD UI and Unit UI can only communicate with their daemons using TCP sockets, but the daemons are isolated in the namespaces and running any GUI inside is tedious and maybe impossible (due to X11/Wayland using TCP sockets and residing outside the namespace) this is why we used *socat* utility to forward TCP sockets in the host machine to Unix sockets that are exposed by the daemons. Because Unix sockets are [virtual] files, and the namespaces are just network namespaces, the Unix sockets are not isolated inside the namespaces, so we could forward them into TCP sockets for the UI. This helped us avoid changing the interface for the UI just to test inside the testbed.

Figure 5.3 shows how the socket forwarding happens with the testbed.

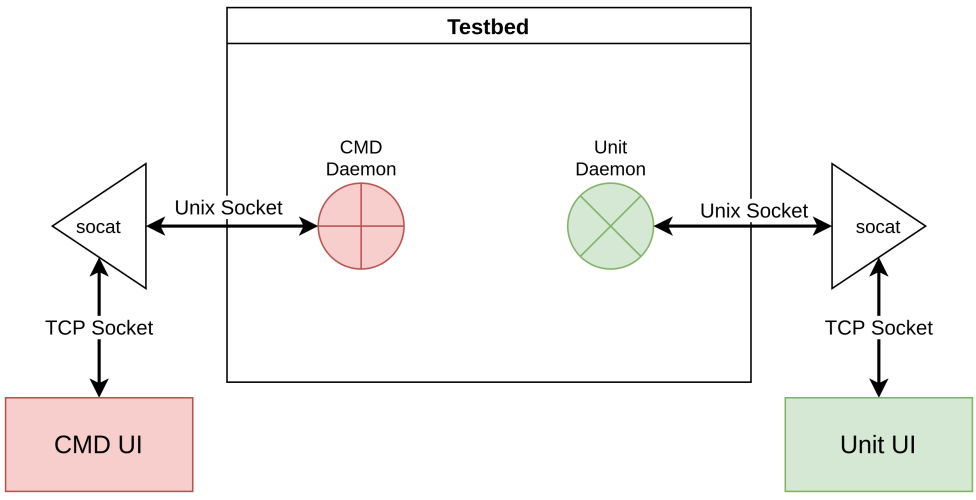

Figure 5.3: Socat Forwarding and UI

5.2 Testing Plan and Strategy

Our main goal in evaluating each module in C4IAN was to make sure it met all functional and non-functional requirements that were specified earlier. Testing is almost as challenging as development, if not more difficult in the world of networks, especially ad-hoc networks. We were very cautious from the very beginning of this project to provide a testing environment

that allowed us to test each and every stage of the development process. For example, the sARP protocol was one of the first implemented protocols related to the Zone-based Hierarchical Link State (ZHLS) unicast protocol; at this stage, we must ensure that the protocol is functioning properly, that each node can identify its neighbors, and that the delay metric to each of them is correctly measured.

For the reasons stated above, testing development could not be postponed after code development. As a result, our testing strategy was obvious from the start in terms of the need for a testing environment that allows us to construct multiple topologies with varied patterns of movement and exchange and route messages between different nodes in these topologies. C4IAN testing development plan can be mainly divided into three components as shown in Figure 5.4: utils testing development, integration testing development and network testing development that serves both module and integration testing.

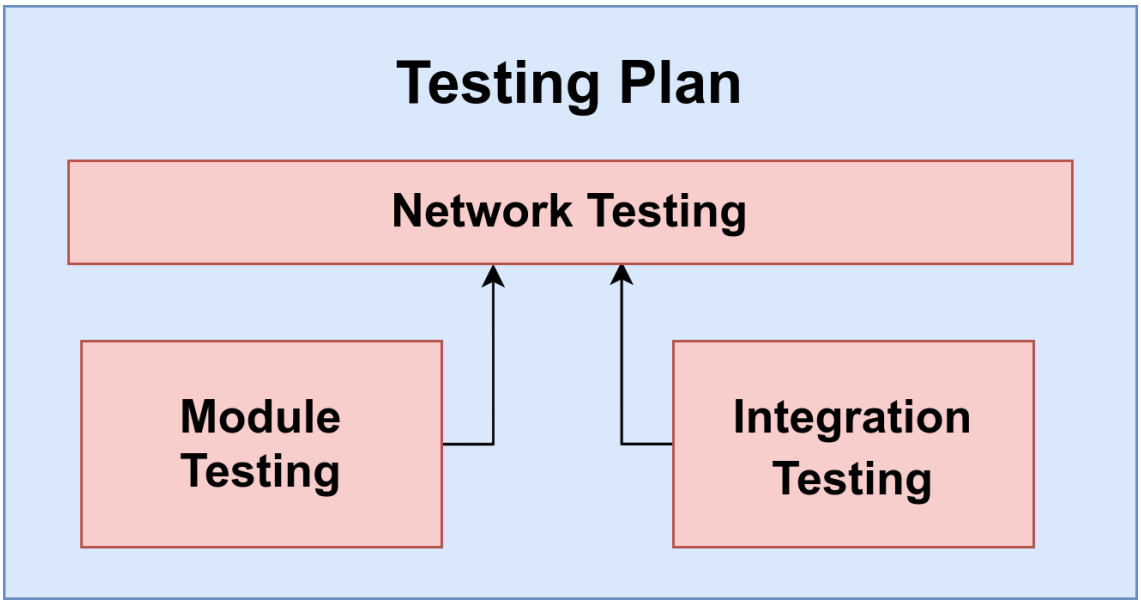

Figure 5.4: Testing Strategy

Each component will be briefly discussed in the section below.

5.2.1 Module Testing

In C4IAN, Module testing is done sequentially into two steps as show in Figure 5.5: utils testing and network testing. All protocols that need an active topology with several nodes, as well as the ability to relocate these nodes and encapsulate each in its own network namespace, fall under the network testing umbrella, and this type of testing development starts from the very beginning of our project. However, utils testing was concerned with aspects that could be verified independently of the network, such as encoding, decoding, marshalling, unmarshalling, graph algorithms, tables handling, and timers. This type of testing was developed immediately after the development of its related code.

5.2.2 Integration Testing

Integration testing was a difficult process in C4IAN. It contains end-to-end testing using a topology that has numerous of nodes, each with its own user interface and router.

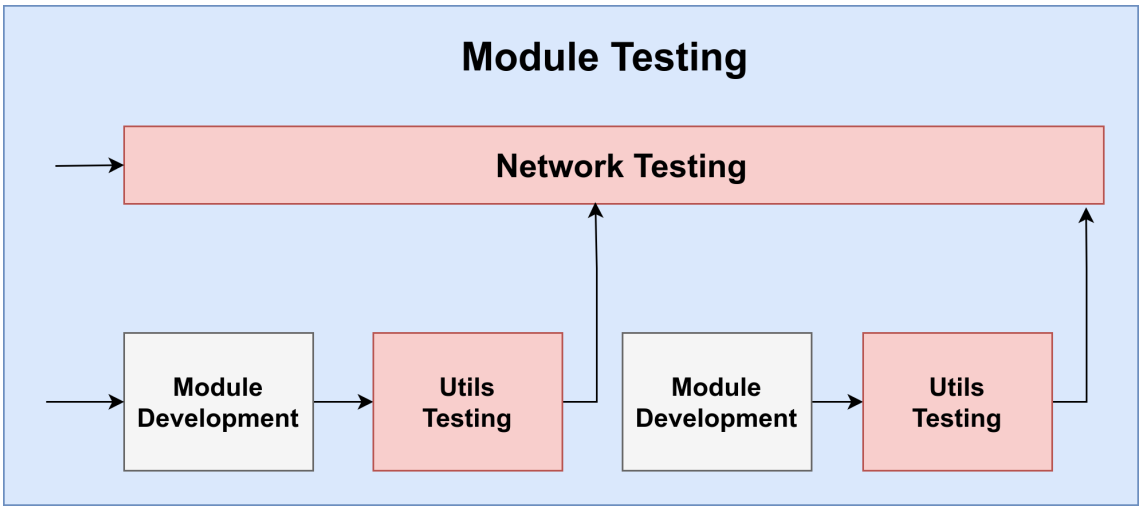

Figure 5.5: Module Testing

The integration testing was split into two primary procedures that ran simultaneously. The initial step was to evaluate the app's functionality from units to command centers and back. This process was carried out both automatically using HAL scripts to simulate units and send periodic sensor data and stream videos to the command center, as well as manually to test both unit and command center user interfaces and ensure that multicast and broadcast data was sent only to its intended destination.

The second step in the integration testing process was to test the routing decisions. Data delivery to its intended destination wasn't enough for us to declare that the system functioned as it's intended to be. It was critical to observe and evaluate each node's routing choice, as well as how data is delivered from its source to its intended destination. To accomplish so, we created a log visualizer software as shown in Figure 5.6 that can collect all the router logs and evaluate their forwarding decisions, and displaying them to us in a way that allows us to appraise the router's performance and behavior.

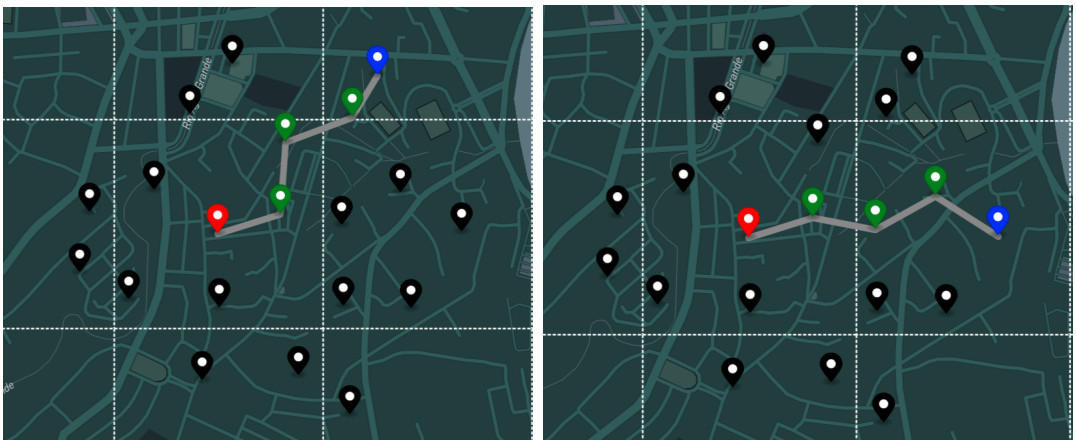

Figure 5.6: Log Visualizer

5.3 Comparative Results to Previous Work

Table 5.1 shows the difference between our work and previous work. We introduced new concepts and extensions to the protocols and implemented them. All the extensions have been discussed in great detail throughout the book.

Protocol	Comparison	Previous Work	Our Work
ZHLS	Zone Size Maintained Topologies	Constant and Predefined Two Separate	Variable Single Unified
Broadcast	Limits	Unbounded	Geographically Limited with a Radius
ODMRP	Join Reply	Using Flooding	Iterating Over Next Hops

Table 5.1: Summary of Differences Between Our Work and Previous Work

Chapter 6

Conclusions and Future Work

We have introduced C4IAN as a communications solution for tactical teams, and thoroughly explained how it works. In this chapter, we let you catch a glimpse of how our journey looked like. We describe the challenges that we faced, the lessons that we learned, and summarize what we achieved. Finally, we discuss what the future may look like for C4IAN.

6.1 Faced Challenges

The core of C4IAN is the router. Routing in MANETs is a research problem that has no established solutions. A multitude of routing protocols are being researched, with only a few implementations. Working on such a problem has introduced us to a variety of challenges.

6.1.1 MANET routing

Routing in MANETs is an open-ended research problem that we had no background of. We needed to accumulate the necessary background knowledge through research papers and books. Afterward, we had to choose and tune adequate routing protocols for our project. We began our journey through literature, understanding and comparing different approaches until we were able to select candidate protocols. Moreover, The papers explaining routing protocols focused on theory. As no open-source implementations exist, we had to fill in the missing pieces during our implementation.

6.1.2 Interfacing with Linux Kernel

The router module works in layer 3 of the TCP/IP network stack. Ideally, it would be implemented as a kernel module that replaces IP routing in Linux. This approach comes with a huge implementation overhead of learning kernel modules, debugging the kernel, and others too many to list. To stay focused on the routing problem, we decided to run the router in user mode. In order to be able to implement routing in user mode, we needed to intercept and re-inject packets at different layers of the Linux network stack. This involved understanding different interfaces offered by the Linux network stack, and a few hacks and tricks to be able to achieve the necessary task in user mode.

6.1.3 Testbed

We decided not to use a simulation for our project, to implement a fully-functional product. However, we still needed a testbed that runs multiple nodes with different topologies and mobility models. Creating the testbed involved writing complex scripts to run multiple instances of the project in separate network namespaces and be able to communicate with them from the default namespace to run the user interfaces, and using a tool called Mininet-WiFi that provides emulated wireless interfaces to simulate connectivity between nodes. Mininet-WiFi was one of a very few available options, so we had to adapt our needs to the tool and its shortcomings.

6.1.4 Testing and Debugging

Testing and debugging the project is not easy, as we are debugging a whole network at once. Even with a small-sized network, troubleshooting involves running several instances at once and tracing their logs concurrently (especially when debugging multicasting). Furthermore, we had to develop a simple GUI that renders the nodes as they move through different zones on a map, to help us create and debug different topologies.

6.1.5 Performance Metrics

C4IAN is implemented as a fully functional program in user space. Other implementations of research papers are only implemented in simulation, while the few publicly available fully functional implementations run in kernel space, and without encryption. This fact made acquiring any performance metrics and testing against other routing solutions almost impossible. The lack of open source implementations of MANET routing solutions combined with the lack of proper tools to emulate large enough mobile ad-hoc networks for testing is disappointing.

6.2 Gained Experience

Working on developing C4IAN has been an enriching experience that we learned from in different ways:

- We gained a deeper understanding of networks and routing, especially MANETs.
- Exposure to research papers and being up-to-date with the advancements in MANET routing.
- Reasoning about protocols and choosing the practical ones, keeping trade-offs in mind.
- Navigating and understanding protocols RFCs.
- Writing highly concurrent programs.
- More experience with low-level implementations and interfacing with Linux.
- We learned several tools and technologies such as *Golang* and *Electron.js*.
- Teamwork, effective collaboration, and asynchronous online communications.

6.3 Conclusions

Routing in MANETs is a research problem that is yet to have solid implementations. C4IAN as a system implements unicast routing, multicast routing, and broadcast routing. It is one of a very few concrete open source implementations. It offers reliable communications without infrastructure. Nonetheless, it does have its limitations. Due to the lack of proper tools, C4IAN is yet to be tested on a very large scale. Introducing scale introduces problems, and stress testing C4IAN in a very large network is something that we do not have the proper tools to do at the moment. Moreover, C4IAN's current implementation only runs on Linux, It is not a cross-platform solution.

6.4 Future Work

C4IAN is one of a few of its kind. It can be extended and built upon in many ways in the future. In this section, we list a few ways in which C4IAN can be extended.

6.4.1 Hardware Deployment

Currently, C4IAN is a software system that provides communications to tactical teams. The system can be deployed on any Linux-based platform for more realistic testing and benchmarking. Moreover, the router can be used for any other application involving MANETs. The system can be extended with proper software clients to be used for communication in fleets of autonomous vehicles or robots.

6.4.2 Multicast Support for TCP

One possible large-scale extension is to build a multicast variation of TCP. Since the scope of our project was concerned with routing, we did not have time to work on such a transport layer protocol. Multicast support for TCP is also a research problem, with no deployed protocols on Linux. The lack of support for TCP in multicasting has restricted our ability to make the most out of multicast routing. If implemented, the use cases of C4IAN can be extended to support reliable delivery of messages and audio to multiple units at once.

6.4.3 Alternatives to Mininet-WiFi

Mininet-WiFi is a great tool that enabled us to test our system with a variety of topologies. Nonetheless, it has its shortcomings. Mininet-WiFi limits the number of nodes that we can use to 100 nodes, which can be a problem when testing for scalability. Furthermore, the way Mininet-WiFi handles mobility and node ranges is a little cumbersome. At the time of writing, we did not find any better alternatives. If a better alternative comes up in the future, it may be worth switching the testbed to use it.

We might need to write our own alternative. At the end, Mininet-Wifi is a wrapper around *wmediumd* and *mac80211_hwsim* which are simple enough to be improved for better networks emulation and for support for more virtual radios.

6.4.4 Uniform Shapes for Hierarchical Zones

In C4IAN, zones are hierarchical and have a rectangular shape. Using more uniform shapes such triangles and hexagons will make zones at the same level have more consistent areas. However, the zone computations will be less straightforward and may be more CPU intensive, which could hurt the limited battery capacity in the handheld devices.

We need to explore the tradeoffs in this area.

6.4.5 Implementing The Router as a Kernel Module

A kernel module is a piece of software that runs in the kernel, it can call any kernel functions and access its data directly with the highest privileges in the system. The advantage of writing a kernel module for the router is that kernel modules don't cause overheads while accessing packets and modifying them. Because they are in the kernel side, they can access the packet buffer directly and modify it in place, which means it takes less time than doing the same in the user space, as the later needs copying from kernel space to user space and back again after the router finishes the packet task.

On the other hand kernel modules are not safe, because if they caused a failure, you won't simply restart it, you might have to restart the whole machine, and maybe even enter a broken state of the whole system where nothing functions and it requires a manual intervention, which is not ideal for a software that might be updated very rarely and be required to function for the most number of continues hours and be very reliable.

Also, if the kernel module has a small bug, it might be a big security hole and might give the attacker a control over the whole system silently. On the other hand, the same bug on the user space might be abused for a small limit. Thus, writing programs for the user space is much safer and secure.

This is a trade-off between performance and security. And we chose security and safety.

Nevertheless, the performance hit, due to copying back and forth, might be small. But we need to assure that by writing a small part of the router in the kernel space and compare the differences. At the end, we might not need to write the whole router in the kernel. We may just write the part that encrypts the packet and adds *ZID* and let it copy the packet's header to user space to decide its forwarding destination, this might reduce the performance penalty.

Doing reactive routing with Linux's forwarding tables is tricky, we also need to explore how to deal with this without user space router.

6.4.6 Forwarding with eBPF

Writing a kernel module is risky because it may damage the whole system for one small bug. Kernel modules are not isolated, and if compromised, the whole system is compromised. Thus *eBPF* is introduced.

Extended Berkeley Packet Filter (eBPF), which found on most *UNIX* operating systems and developed lately for Microsoft Windows. It's an interface for user space programs to insert code that runs safely in the kernel. This code can filter syscalls, packets and events on the kernel and operate on those data. The safe part of eBPF is that all code is executed in a virtual machine, and before executing them they are verified to not access any unauthorized data or cause any memory leaks or destroy any memory it doesn't hold.

eBPF may speed up forwarding as it can just operate on the kernel space directly and thus safe time of copying packets back and forth. Also, they are generally much safer than writing kernel modules directly, thanks to the verification system and the virtual machine.

BPF was designed first for networking applications, and later came the extensions which serve other applications, like security and syscalls modifications, that seek high performance integration with the kernel functionality.

We might need to explore using eBPF for the forwarding plan on the performance of the router.

6.4.7 No FFmpeg

Currently, we use *FFmpeg* for video streaming. We give it the video source, let it be a camera stream or a video file or a URL for another stream, and it cuts it into HLS chunks and produces `.m3u8` metadata files which we move around to the frontend player.

We may reach a better user experience by reducing the complexity of the code base, and instead of calling another program, which has its downsides of how to deal with evens back and forth between the producer and consumer of the Http Live Streaming (HLS) segments which we fixed in a non-ideal way. We might implement our own HLS segmenter, which just segment the video source in memory directly and serve with minimal latency.

6.4.8 Different Quality Levels of Streaming

Currently, the unit daemon only streams at one level of quality, this is for simplicity. But we might need to have multiple available streams with different qualities for each one. HLS already supports this, but the problem resides in the power and memory tradeoffs against the performance, given that the unit devices have limited capacity for battery.

We might need to try a different approach than the one HLS took. If the network and battery are in good conditions, the CMD daemon gets notified of the availability of streaming at a higher quality, and they may select a higher quality if they wanted to which the unit reacts to by encoding and streaming at the selected quality level.

This is different from what HLS does, which is making all the levels available proactively. Clearly, HLS' approach is not suitable for handheld devices, and we need a reactive approach. This may require us to extend the protocol.

Bibliography

[1] Abdul Hadi Abd Rahman and Zuriati Ahmad Zukarnain. "Performance comparison of AODV, DSDV and I-DSDV routing protocols in mobile ad hoc networks". In: *European Journal of Scientific Research* 31.4 (2009), pp. 556–576.

[2] Arpit Bansal et al. "Performance Analysis of ZHLS-GF Routing Protocol for MANETs through simulations". In: *Research Cell: An International Journal of Engineering Sciences ISSN* (2011), pp. 2229–6913.

[3] Nicklas Beijar. "Zone routing protocol (ZRP)". In: *Networking Laboratory, Helsinki University of Technology, Finland* 9 (2002), pp. 1–12.

[4] Gyanappa A. Walikar Rajashekar C. Biradar. "A survey on hybrid routing mechanisms in mobile ad hoc networks". In: *Journal of Network and Computer Applications* 77.1 (2016), pp. 48–63. DOI: https://doi.org/10.1016/j.jnca.2016.10.014.

[5] *Bittium Website Kernel Description*. https://www.bittium.com. Accessed: 2021-07-23.

[6] Ian D Chakeres and Elizabeth M Belding-Royer. "AODV routing protocol implementation design". In: *24th International Conference on Distributed Computing Systems Workshops, 2004. Proceedings.* IEEE. 2004, pp. 698–703.

[7] Chee-Onn Chow and Hiroshi Ishii. "Multiple tree multicast ad hoc on-demand distance vector (mt-maodv) routing protocol for video multicast over mobile ad hoc networks". In: *IEICE transactions on communications* 91.2 (2008), pp. 428–436.

[8] Thomas Clausen and Philippe Jacquet. *Rfc3626: Optimized link state routing protocol (olsr)*. 2003.

[9] Thomas Clausen et al. "Optimized link state routing protocol (OLSR)". In: (2003).

[10] Naveen Garg, Kiran Aswal, and Dinesh C Dobhal. "A review of routing protocols in mobile ad hoc networks". In: *International Journal of Information Technology and Knowledge Management January-June* 5 (2012), pp. 177–180.

[11] Min Ge, Srikanth V. Krishnamurthy, and Michalis Faloutsos. "Application versus network layer multicasting in ad hoc networks: the ALMA routing protocol". In: *Ad Hoc Networks* 4.2 (2006), pp. 283–300. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2004.10.002. URL: https://www.sciencedirect.com/science/article/pii/S1570870504000885.

[12] *Hytera Finance*. https://www.wsj.com/market-data/quotes/CN/002583/financials. Accessed: 2021-07-23.

[13] Mario Joa-Ng and I-Tai Lu. "A peer-to-peer zone-based two-level link state routing for mobile ad hoc networks". In: *IEEE Journal on selected areas in communications* 17.8 (1999), pp. 1415–1425.

[14] David B Johnson, David A Maltz, Josh Broch, et al. "DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks". In: *Ad hoc networking* 5.1 (2001), pp. 139–172.

[15] Luo Junhai, Xue Liu, and Ye Danxia. "Research on multicast routing protocols for mobile ad-hoc networks". In: *Computer Networks* 52.5 (2008), pp. 988–997.

[16] Luo Junhai et al. "A survey of multicast routing protocols for mobile ad-hoc networks". In: *IEEE communications surveys & tutorials* 11.1 (2009), pp. 78–91.

[17] Satoshi Kamidate et al. "Fault-tolerant ZHLS routing protocol for wireless mobile ad hoc network". In: *Proceedings of the 7th WSEAS International Conference on Multimedia, Internet & Video Technologies*. Citeseer. 2007.

[18] Sung-Ju Lee, M. Gerla, and Ching-Chuan Chiang. "On-demand multicast routing protocol". In: *WCNC. 1999 IEEE Wireless Communications and Networking Conference (Cat. No.99TH8466)*. Vol. 3. 1999, 1298–1302 vol.3. DOI: 10.1109/WCNC.1999.796947.

[19] Prasant Mohapatra and Srikanth Krishnamurthy. *AD HOC NETWORKS: technologies and protocols*. Springer Science & Business Media, 2004.

[20] *MS Windows NT Kernel Description*. https://www.hytera.com. Accessed: 2021-07-23.

[21] M. Natkaniec and A.R. Pach. "PUMA - a new channel access protocol for wireless LANs". In: *The 5th International Symposium on Wireless Personal Multimedia Communications*. Vol. 3. 2002, 1351–1355 vol.3. DOI: 10.1109/WPMC.2002.1088400.

[22] Soon Y. Oh, Joon-Sang Park, and Mario Gerla. "E-ODMRP: Enhanced ODMRP with motion adaptive refresh". In: *Journal of Parallel and Distributed Computing* 68.8 (2008), pp. 1044–1053. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2008.04.004. URL: https://www.sciencedirect.com/science/article/pii/S0743731508000646.

[23] Dhammika Pathirana and Minseok Kwon. "RODMRP: Resilient On-Demand Multicast Routing Protocol". In: *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*. Vol. 2. 2007, pp. 85–92. DOI: 10.1109/AINAW.2007.315.

[24] Francesco Saverio Proto and Claudio Pisa. "Implementation of the OBAMP overlay protocol for multicast delivery in OLSR wireless community networks". In: *2010 IEEE International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. 2010, pp. 1–3. DOI: 10.1109/WOWMOM.2010.5534941.

[25] Ram Ramanathan and Jason Redi. "A brief overview of ad hoc networks: challenges and directions". In: *IEEE communications Magazine* 40.5 (2002), pp. 20–22.

[26] Roberto Baldoni Roberto Beraldi. "Unicast Routing Techniques for Mobile Ad Hoc Networks". In: (2003). DOI: https://doi.org/10.1201/9781420040401.ch7.

[27] Elizabeth M Royer and Charles E Perkins. "An implementation study of the AODV routing protocol". In: *2000 IEEE Wireless Communications and Networking Conference. Conference Record (Cat. No. 00TH8540)*. Vol. 3. IEEE. 2000, pp. 1003–1008.

[28] S Sathish, K Thangavel, and S Boopathi. "Comparative analysis of DSR, FSR and ZRP routing protocols in MANET". In: *International Conference on Information and Network Technology IPCSIT vol*. Vol. 4. 2011.

[29] A Suruliandi and Sam Pradeep Raj. "A SURVEY ON MULTICAST ROUTING PROTOCOLS FOR PERFORMANCE EVALUATION IN WIRELESS SENSOR NETWORK". In: *INTERNATIONAL JOURNAL ON COMMUNICATION TECHNOLOGY* 6 (Mar. 2015), pp. 2229–6948. DOI: 10.21917/ijct.2015.0153.

[30] T. Hamma T. Katoh B.B. Bista T. Takata. "An Efficient ZHLS Routing Protocol for Mobile Ad Hoc Networks". In: *International Workshop on Database and Expert Systems Applications* (2006). DOI: https://doi.org/10.1109/DEXA.2006.24.

[31] Nor Surayati Mohamad Usop, Azizol Abdullah, Ahmad Faisal Amri Abidin, et al. "Performance evaluation of AODV, DSDV & DSR routing protocol in grid environment". In: *IJCSNS International Journal of Computer Science and Network Security* 9.7 (2009), pp. 261–268.

[32] Mladen Đuro Veinović. "Comparative analysis of unicast routing protocols in MANET networks". In: *Synthesis* (2015). DOI: https://doi.org/10.15308/Synthesis-2015-112-115.

[33] Jie Wu and Ivan Stojmenovic. "Ad hoc networks". In: *Computer* 37.2 (2004), pp. 29–31.

Appendices

Appendix A

Development Platforms and Tools

A.1 Hardware Tools

Personal computers running linux-based oberating systems.

A.2 Software Tools

A.2.1 Programming Languages

A.2.1.1 Golang

Go was created at Google to help programmers be more productive in an era of multicore, networked devices by including the following features:

1. Run-time efficiency (like C).
2. Multiprocessing and high-performance networking.
3. Readability and usability are two of the most important factors to consider (like Python).

Usage: Main programming to build the core modules and daemons.

A.2.1.2 Python

Python is a high-level programming language that is interpreted and object-oriented. It's ideal for Rapid Application Development, as well as using as a scripting or glue language to link together existing components.
Usage: To write testbed scripts

A.2.1.3 Shell Scripting

Shell scripting is a type of scripting that runs in the Unix/Linux shell. It can condense long and repeated command sequences into a single, easy-to-understand script.
Usage: For running repetitive commands with a single script.

A.2.2 Libraries and Frameworks

A.2.2.1 Mininet

Mininet builds a realistic virtual network on a single system, running actual kernel, switch, and application code.
Usage: For testing our system with a variety of topologies.

A.2.2.2 Socat

Socat is a command-line utility for creating and transferring data across two bidirectional byte streams.
Usage: For forwarding data from TCP to Unix sockets to make the user interface works on the browser, however the daemons are running inside containers without changing the interfacing.

A.2.2.3 Nsenter

It's a simple utility that lets you enter namespaces. It can technically enter existing namespaces or start a new process in a different set of namespaces.
Usage: For Running the programs inside containers.

A.2.2.4 React.js

React is a JavaScript library for creating user interfaces that is fast and adaptable.
Usage: Build both Command Center and Unit user interface (UI) as a web application.

A.2.2.5 Electron.js

Electron.js is a runtime framework for developing desktop applications.
Usage: Build both Command Center and Unit user interface (UI) as a desktop application.

A.2.3 Tools and Platforms

- Visual Studio Code: it is a free source-code editor with debugging support.
- GitHub: it is an online service that hosts Git repositories for software development and version control.

Appendix B

Use Cases

C4IAN is designed to provide infrastructure-less wireless communications for tactical teams. It provides the following services for command centers and units, who can be moving freely:

- Exchanging code messages between units and command centers.
- Exchanging audio messages between units and command centers.
- Assigning units to groups, and sending audio messages from command centers to a specific groups (multicasting), or to all units (broadcasting).
- Streaming sensors' data (location, heartbeat, etc.) from units to command centers.
- Streaming videos from combat cameras at the units to command centers on-demand.
- Recording and visualizing all communications history with different units at command centers.

These services are provided by the implemented clients for command centers and units. However, the underlying router is a generic MANET router that can be easily used for further applications by modifying or extending the software clients. Some examples are:

- Controlling fleets of drones or autonomous vehicles.
- Providing communications for smart robots with decentralized coordination.
- Aggregating sensors data from moving vehicles.

Appendix C

User Guide

This guide is for users of C4IAN who want to setup the environment and run the applications. In this guide, we won't write about the modules themselves or their functions, we expect that you have read chapter 4 and 5.

This guide is divided into 2 sections:

- Environment Setup: The dependencies and scripts that are needed to deploy the system.
- System Deployment: How to deploy each module (Router, Unit, CMD) in production.
- Usage: How to navigate the UI for both the CMD and Unit.

C.1 Environment Setup

You need to install the following packages on your system:

- yarn
- NodeJS
- socat
- python3
- python3-pip
- a web browser (chrome, chromium or Firefox)

You may need to consult your system's manual on how to install them, because it differs from a distro to another.

The rest of this section assumes you are on a Debian derived OS. If it's a different OS, you may need to edit the commands.

When you see $ it means the command should be executed as a shell command.

C.1.1 Golang 1.15.8

```
$ sudo apt update && sudo apt install -y wget && (
    set -e
    cd /tmp
    wget -c https://golang.org/dl/go1.15.8.linux-amd64.tar.gz
    sudo tar xvf go1.15.8.linux-amd64.tar.gz
    sudo chown -R root:root ./go
```

```
    sudo mv ./go /usr/local
    mkdir -p $HOME/.config/go/1.15/{bin,pkg,src}
    echo >>"$HOME"/.bashrc
    echo 'export GOPATH="$HOME/.config/go/1.15"' >>"$HOME"/.bashrc
    echo 'export PATH="$PATH:/usr/local/go/bin:$GOPATH/bin"'\
        >>"$HOME"/.bashrc
    echo >>"$HOME"/.bashrc
    . "$HOME"/.bashrc
)
```

C.1.2 python packages

To install python packages, you should run:

```
$ sudo python3 -m pip install simple-websocket-server numpy watchgod
```

C.1.3 Libnetfilter-queue

Needed for the router to get the packets from the kernel, it maybe installed on your system, here is how to get it:

```
$ sudo apt update && sudo apt install -y libnetfilter-queue-dev
```

C.1.4 Known Issues

C.1.4.1 Electron with libgconf

```
electron: error while loading shared libraries: libgconf-2.so.4: cannot
open shared object file: No such file or directory
```

Solution Run:

```
$ sudo apt install -y libgconf-2-4
```

C.1.4.2 Writing to `/tmp/`

On some systems, writing to `/tmp/` directory may be disabled, we need to write to it for many different reasons. So if some script complained that there are no permissions to write to /tmp/ you may need to execute the folowing command:

```
$ sudo sysctl fs.protected_regular=0
```

C.2 System Deployment

This section is divided into 3 subsections on how to deploy:

- The Router.
- Unit.
- CMD.

You need to execute all the next commands in `src/` directory.

Before running any program you need to build them all, run:

```
$ make apps router
```

C.2.0.1 Router Deployment

Router is deployed on all devices that should communicate within an adhoc network, whether its unit or cmd device.

To enter adhoc mode and run the router locally assuming interface=wlan0 and ip=10.0.0.1

```
$ sudo ./start local router -- wlan0 10.0.0.1
```

or to override default env variables assuming password is `somePass` and the `SSID` you want to use is `someSSIDName`:

```
$ sudo PASS=somePass SSID=someSSIDName ./start local router -- wlan0 10.0.0.1
```

You may need to pass more options to the router to control it more. To get a list of router's arguments:

```
$ ./router/router --help
```

C.2.0.2 Unit Deployment

Unit is only deployed on unit devices alongside the router.

You might need to develop a `HAL` program that interfaces with the unit daemon. The interface for HAL is at `src/unit/halapi` with an example implementation (for testing on emulation) in `src/unit/halsimulation`.

To deploy the unit daemon locally, simply run:

```
$ sudo ./start local unit
```

You may need to pass more options to the unit daemon to control it more. To get a list of units's arguments:

```
$ ./unit/daemon/daemon --help
```

C.2.0.3 Command-Center Daemon Deployment

Command-Center is only deployed on command center computers alongside the router.

To deploy the cmd daemon locally, simply run:

```
$ sudo ./start local cmd
```

You may need to pass more options to the cmd daemon to control it more. To get a list of units's arguments:

```
$ ./cmd/daemon/daemon --help
```

C.2.0.4 Command-Center UI Deployment

One command center daemon may be connected to multiple UI processes. They could be deployed from multiple computers.

To run the cmd UI locally, run:

```
$ cd cmd/ui && yarn && yarn run electron
```

C.3 Usage

This section describes how to use the system modules.

C.3.1 Router

For the router, no changes should be made on the system to use it. Every program shall proceed as normal with opening TCP/UDP sockets and communicating with the outside world as usual. Note that the router doesn't route Unix sockets traffic, only TCP/UDP/ICMP over IPv4.

C.3.2 Unit

It depends on what HW you are building. If you used the unit UI, it just functions as the cmd UI, which we will describe in the subsection C.3.3. If you used the halsimulation, it just sends commands and messages, and you don't interact with it.

C.3.3 Command Center UI

As shown in Figure C.1 You should have a window with the UI asking you about the port of the daemon, to get it consult the logs of the running cmd daemon.

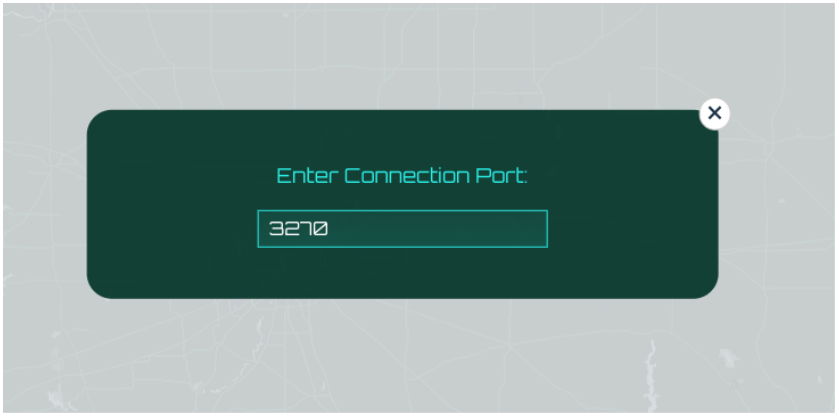

Figure C.1: CMD UI Filling Port Number

After setting the port, you will be directed to the home page (Figure C.2) which shows a map of units and a menu (Figure C.3) and a floating button if clicked shows the button for broadcasting a message (Figure C.4 and Figure C.5.)

Clicking on the *Units* from menu button directs to the units page, which contains the list of units and videos and audios and code messages, as shown in Figure C.6.

Figure C.2: CMD UI Map Page

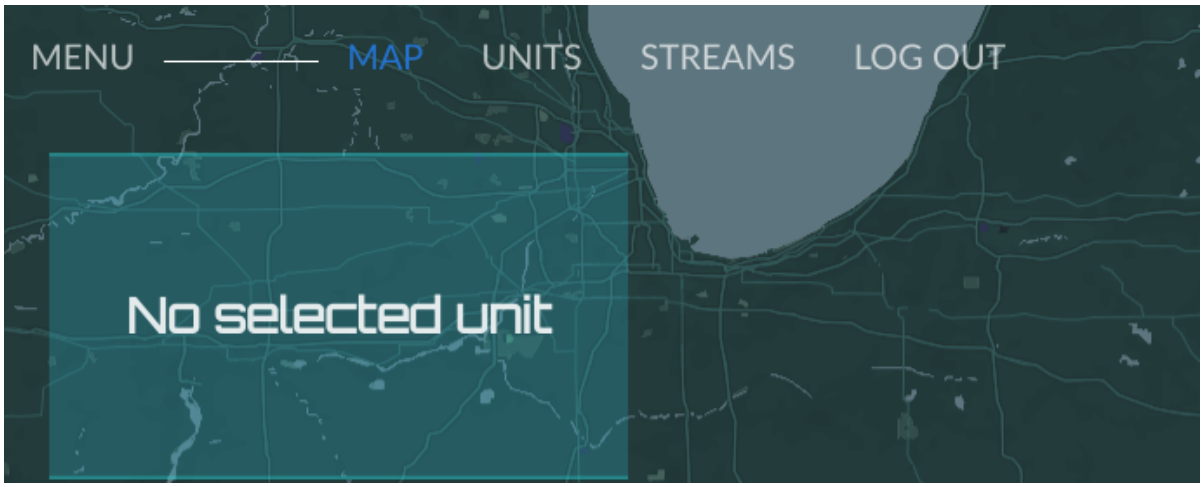

Figure C.3: CMD UI Menu

Figure C.7 shows how to send a code message by clicking on one of the icons, each has its code, which will be sent to the selected unit.

You may send a video streaming request to the unit, as shown in Figure C.8. And then you need to go to the streaming page, from the menu button, as show in Figure C.9, which takes you to a page showing the video streaming live from the unit's video sources with the captured audio (if any).

Figure C.4: CMD UI Broadcast Button

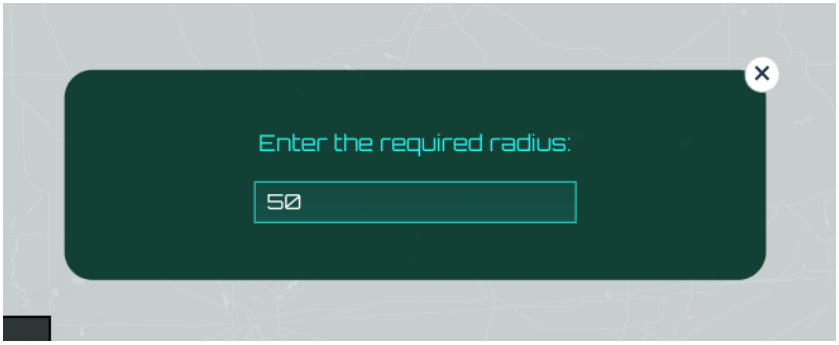

Figure C.5: CMD UI Broadcast Dialog

Figure C.6: CMD UI Units Page

Figure C.7: CMD UI Code Message (Attack)

Figure C.8: CMD UI Requesting Video Streaming From Unit

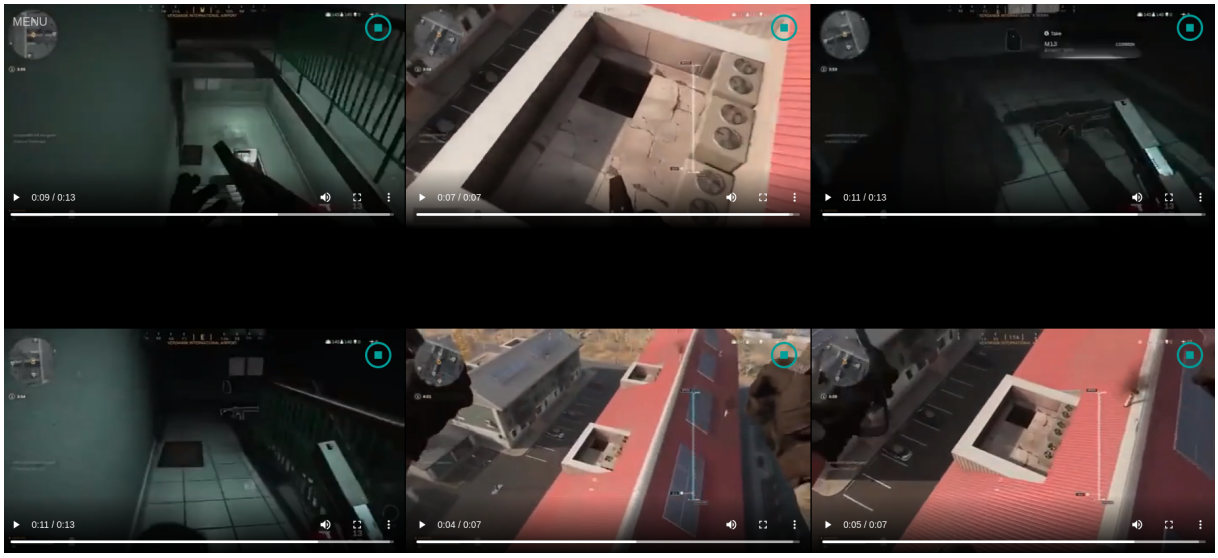

Figure C.9: CMD UI Streaming Page

Appendix D

Feasibility Study

In this appendix, we discuss the feasibility of C4IAN. As chapter 2 covered marketing and financial aspects, we will focus on technical, operational, and legal feasibility.

D.1 Technical Feasibility

To design C4IAN, a strong background in networks is required, especially mobile ad-hoc networks. The resources for learning networks are abundant, but those for mobile ad-hoc networks are scarce. Learning about routing in mobile ad-hoc networks and choosing adequate protocols to implement is not an easy task.

After the produce is designed, the implementation phase requires skills in low level programming. As the router module interfaces extensively with the Linux kernel, and it's network stack, knowledge about low level programming and networks programming is a must. Since the Linux kernel is open source, the implementation could be completed without major obstacles. The router could be implemented in user mode, or as a kernel module. Both are available options due to the flexibility of the Linux kernel. As Linux is a relatively lightweight operating system that can be deployed on various hardware, deploying C4IAN on most types of hardware should not be problematic.

D.2 Operational Feasibility

From the user's point of view, C4IAN is a very easy system to operate. Most of the complexity is hidden in the router implementation, which the users (and the applications) do not directly interact with. The router module interacts exclusively with the Linux kernel. From the perspective of a user, or an application developer, the system is transparent. The implementation of applications is not affected by the design or implementation of the router. This gives a lot of flexibility for developers working on producing applications that utilize the routing capabilities offered by C4IAN. As for the command center and unit applications provided by C4IAN, they are very straightforward to deal with. They provide a simple interface for users to send and receive messages through simple GUIs. This is explained further in C.3.

D.3 Legal Feasibility

C4IAN strictly uses open source packages in its development, so the usage and distribution of C4IAN and its code should not cause any legal troubles.

In our initial design of C4IAN, we use Wi-Fi as the MAC Layer. This causes no legal problems as Wi-Fi bands belong to the ISM bands, which are free to use without licensing. For some applications, a different band with different properties may be required. Even though the router implementation mostly will not need to be modified to accommodate other MAC layers, some legal obstacles may rise. Using bands outside the ISM bands requires legal licensing, with laws differing by country. If C4IAN is to be operated in such bands, necessary licensing must be acquired first.

D.4 SWOT Analysis

Figure D.1 shows the SWOT (Strengths, Weaknesses, Opportunities and Threats) analysis of the product. SWOT analysis helps in project and expectations management.

One of the main weaknesses of the product is that it only targets Linux. However, it is not a major problem, as at the end the product will be installed on closed embedded systems and mostly users will be indifferent.

The product is user-friendly and scalable, yet the customers have difficult requirements.

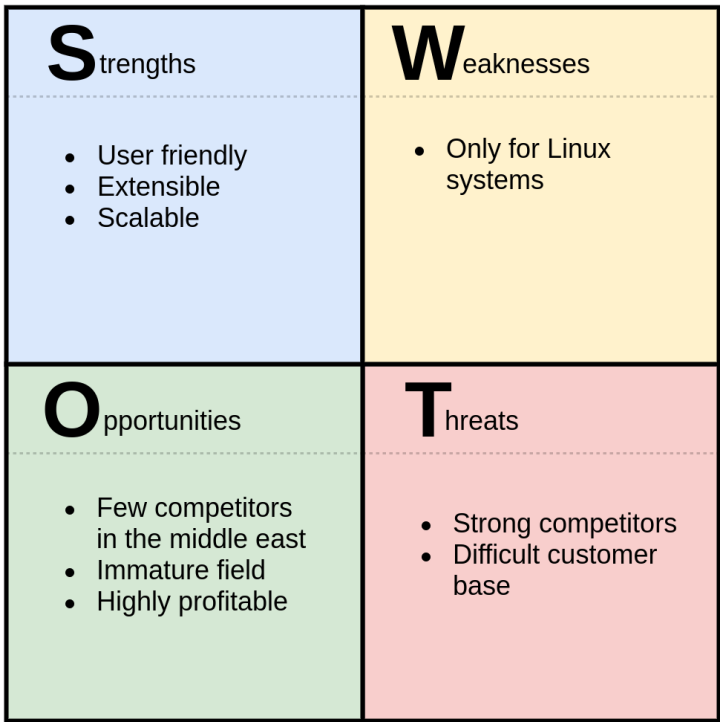

Figure D.1: SWOT Analysis