SOFTWARE-DEFINED ARCHITECTURE AND ROUTING SOLUTIONS FOR MOBILE AD HOC NETWORKS

by

Ayush Dusia

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Fall 2019

SOFTWARE-DEFINED ARCHITECTURE AND ROUTING SOLUTIONS FOR MOBILE AD HOC NETWORKS

by

Ayush Dusia

Approved: _____
Kathleen F. McCoy, Ph.D.
Chair of the Department of Computer and Information Sciences

Approved: _____
Levi T. Thompson, Ph.D.
Dean of the College of Engineering

Approved: _____
Douglas J. Doren, Ph.D.
Interim Vice Provost for Graduate and Professional Education and
Dean of the Graduate College

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Adarshpal S. Sethi, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Chien-Chung Shen, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Lena Mashayekhy, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Vinod K. Mishra, Ph.D.
Member of dissertation committee

ACKNOWLEDGEMENTS

Several individuals have supported and guided me in my pursuit of a Ph.D. degree. I want to thank all of them and mention a few special ones below.

First and foremost, my advisor, Dr. Adarsh Sethi, has been my pillar of strength. Not only has he provided me academic guidance, but he also mentored me into becoming a better person. He is an excellent teacher and advisor, but more importantly, a wonderful person. This dissertation would not have been possible without his support.

The learnings from my experience of working with Dr. Ram Ramanthan have helped me significantly in designing the solutions presented in this dissertation. I will always be thankful for the opportunity of working with him.

I also want to thank my dissertation committee members: Dr. Vinod K. Mishra, Dr. Chien-Chun Shen, and Dr. Lena Mashayekhy, and other faculty members at the University of Delaware for all the enjoyable discussions and guidance over the years.

Special thanks to Dr. Paul Amer for offering me a desk in his lab and introducing me to the research life, and also for encouraging me to enroll in the Ph.D. program. Special thanks also to Dr. Blake Meyers for providing me the opportunity of working in his lab and for supporting me financially for so many years.

I have had so many amazing friends over the years, naming just a few of them in the lexical order - Aman Sawhney, Atul Kakrana, Divya Chintada, Fan Yang, Moumita Bhattacharya, Pradnya Powar, Samir Gupta, Sandeep Rath, Siddhisanket Raskar, and Tianye Ma. I will always be grateful for the memories I have with them.

Lastly, I want to thank my parents and my brother for supporting me at every stage, for encouraging me to keep moving forward, and for guiding me with their experiences. I dedicate this dissertation to my brother, Kunal Dusia, for envisioning all of this, laying the foundation, and paving the way.

TABLE OF CONTENTS

LIST OF TABLES . . . . . **x**
LIST OF FIGURES . . . . . **xii**
ABSTRACT . . . . . **xviii**

Chapter

1 INTRODUCTION **1**

1.1 Software-Defined Networking 2
1.2 Dissertation Contributions 4
1.3 Dissertation Outline 7

2 BACKGROUND AND RELATED WORKS **10**

2.1 Traditional Network Architecture 10
2.2 Software-Defined Networking Architecture 12
2.3 Mobile Ad hoc Networks 15
2.4 SDN-based Architectures for MANETs 17
2.5 Low-Rate Long-Range Networking 19

3 SOFTWARE-DEFINED MOBILE AD HOC NETWORK **22**

3.1 Design Considerations 22
3.2 Architecture 22
3.3 Internal Structure of Node 24
3.4 ns3 Simulator Modifications 26
3.5 SD-MANET Opportunities 27

4 SD-MANET ROUTING **29**

4.1 The PCC Protocol 29

4.1.1 Learning Route To SDNC 29

4.1.2 Learning Network Topology . . . 30
4.1.3 Sending Network Routes . . . 32

4.2 Communication Complexity . . . 34
4.3 Simulation Results . . . 37
4.4 Conclusions . . . 39

5 ECHO . . . 41

5.1 Network-Wide Broadcast . . . 42
5.2 The ECHO Protocol . . . 44

5.2.1 Determining Critical Nodes . . . 45
5.2.2 Managing Full-Flood Generation . . . 46
5.2.3 Overhead . . . 48

5.3 Theoretical Analysis . . . 48
5.4 Communication Complexity . . . 51
5.5 Simulation Results . . . 54

5.5.1 Increasing Network Size . . . 56
5.5.2 Increasing Network Density . . . 57
5.5.3 Increasing Network Load . . . 59
5.5.4 Increasing Node Speed . . . 59
5.5.5 Increasing Network Size (1 Mbps Data Rate) . . . 60

5.6 Conclusions . . . 62

6 VINE . . . 64

6.1 Routing in Low-Power Wide Area Networks (LPWANs) . . . 65
6.2 The VINE Protocol . . . 66

6.2.1 Gradient Establishment . . . 67
6.2.2 Packet Forwarding . . . 70
6.2.3 Discussion: Flooding and Control Information . . . 71
6.2.4 VINE Example . . . 72

6.3 Communication Complexity Analysis . . . 74
6.4 Simulation Results . . . 76

6.4.1 Increasing Network Size . . . 78
6.4.2 Increasing Network Density . . . 80

6.4.3 Increasing Network Load 81
6.4.4 Increasing Network Size (1 Mbps Data Rate) 82

6.5 Conclusions 83

7 CENTRALIZED OPPORTUNISTIC REACTIVE ROUTING . . 85

7.1 The CORR Protocol 86

7.1.1 Learning Route to SDNC 86
7.1.2 Learning Network Topology 88
7.1.3 Sending Network Routes 90

7.1.3.1 Sending RU Messages 91
7.1.3.2 Forwarding Data Packets 91

7.2 Communication Complexity Analysis 94
7.3 Simulation Results 98

7.3.1 Increasing Network Size 99
7.3.2 Decreasing Network Density 101
7.3.3 Increasing Network load 102
7.3.4 Increasing Node Speed 103

7.4 Conclusions 104

8 CENTRALIZED PROACTIVE ROUTING 106

8.1 The CPR Protocol 106

8.1.1 Sending Network Routes 107

8.2 Communication Complexity 108
8.3 Simulation Results 110

8.3.1 Increasing Network Size 112
8.3.2 Decreasing Network Density 113
8.3.3 Increasing Network Load 114
8.3.4 Increasing Node Speed 115

8.4 Scalability Issues 116
8.5 Conclusions 117

9 HIERARCHICAL CENTRALIZED PROACTIVE ROUTING . . 118

9.1 The HCPR Protocol . . . 118

9.1.1 Learning Route to SDNC . . . 119

9.1.1.1 Cluster Formation . . . 119
9.1.1.2 Gateway Nodes . . . 122

9.1.2 Learning Network Topology . . . 122
9.1.3 Sending Network Routes . . . 123

9.1.3.1 Intra-Cluster Routing . . . 123
9.1.3.2 Inter-Cluster Routing . . . 124
9.1.3.3 Data packet Forwarding . . . 126

9.2 Communication Complexity . . . 127
9.3 Simulation Results . . . 131

9.3.1 Increasing Network Size . . . 132
9.3.2 Decreasing Network Density . . . 133
9.3.3 Increasing Network Load . . . 134
9.3.4 Increasing Node Speed . . . 135

9.4 Conclusions . . . 136

10 SUMMARY AND FUTURE DIRECTIONS . . . 137

10.1 Summary . . . 138
10.2 Future Work . . . 141
10.3 Other Research Projects . . . 144

BIBLIOGRAPHY . . . 145

Appendix

SD-MANET CONTROL MESSAGE DESIGN . . . 157

A.1 RFC5 444 Specifications . . . 157

A.1.1 Packet . . . 157

A.1.2 Messages 158

A.1.2.1 Message Header 159
A.1.2.2 Message Body 160

A.1.3 Address Blocks 160
A.1.4 TLV and TLV Block 163

A.2 SD-MANET Control Messages 165

A.2.1 Topology Discovery (TD) 166
A.2.2 Neighbor Information (NI) 168
A.2.3 Route Request (RR) 169
A.2.4 Route Update (RU) 169
A.2.5 Route Update Acknowledgment (RUA) 174
A.2.6 Cluster Information (CI) 174

LIST OF TABLES

4.1 PCC Communication Complexity Symbols 35

4.2 PCC Message Communication Complexity 36

4.3 PCC Asymptotic Control Communication Complexities 37

4.4 PCC Simulation Parameters 37

5.1 ECHO Communication Complexity Symbols 51

5.2 ECHO Asymptotic Communication Complexity 54

5.3 ECHO Simulation Scenarios 55

5.4 ECHO Simulation Parameters 55

6.1 VINE Communication Complexity Symbols 74

6.2 VINE Simulation Scenarios 77

6.3 VINE Simulation Parameters 78

7.1 CORR Communication Complexity Symbols 95

7.2 CORR Simulation Scenarios 98

7.3 CORR Simulation Parameters 99

8.1 CPR Communication Complexity Symbols 108

8.2 CPR Message Communication Complexity 109

8.3 CPR Asymptotic Control Communication Complexities 110

8.4 CPR Simulation Scenarios 111

8.5 CPR Simulation Parameters 111

9.1 HCPR Communication Complexity Symbols 127

9.2 HCPR Message Communication Complexity 128

9.3 HCPR Asymptotic Control Communication Complexities 130

9.4 HCPR Simulation Scenarios 131

9.5 HCPR Simulation Parameters 132

A.1 Interpretations of the **ahasfulltail** and **ahaszerotail flags** 162

A.2 Interpretations of the **ahassingleprelen** and **ahasmultiprelen flags** 162

A.3 Interpretations of the **thassingleindex** and **thasmultiindex flags** 164

A.4 Interpretations of the **thasvalue** and **thasextlen flags** 164

A.5 Interpretations of the **thassingleindex** and **thasmultiindex flags** 165

A.6 SD-MANET Control Messages And Protocols Mapping 166

A.7 Topology Discovery Message Fields 168

A.8 Routing Information Example 1 171

A.9 Routing Information Example 2 172

A.10 Flow Table 173

A.11 Route Update Message Fields 174

LIST OF FIGURES

1.1 Dissertation Outline 7

2.1 Traditional Network Architecture 11

2.2 Software-Defined Networking Architecture 12

2.3 Packet Matching Fields 13

2.4 SDN Controller 14

3.1 Software-Defined Mobile Ad hoc Network Architecture 23

3.2 Internal Structure Of SD-MANET Node 25

3.3 NS3 Modifications 27

4.1 SDNC flooding a Topology Discovery (TD) message and nodes A, B, C, and D learning their route to SDNC. 31

4.2 Node D sending a Neighbor Information (NI) message to SDNC via its RTS. 32

4.3 SDNC sending a Route Update (RU) message to node D. 33

4.4 Node D sending a Route Update Acknowledgment (RUA) message to SDNC via the RTS. 34

4.5 Results showing Total Control Messages and Routing Overhead. . . 38

4.6 Results showing Packet Delivery Ratio and Average Delay. 39

5.1 ECHO picks a random packet periodically to flood (Full Flood). This data packet is used to select critical nodes. After that, only critical nodes relay the data packets (Pruned Flood). In this example, node A originates a data packet. This is marked FF and flooded. All nodes use Algorithm 4 (see Section 5.2.1) to compute critical node. Thereafter, all packets are relayed only by critical nodes 45

5.2 ECHO critical node computation state diagram. "Echo Received" means receiving a Full Flood with previous-sender marked as own id. 46

5.3 Example ECHO operation on FF originating from node A. Since nodes A and C are the only nodes that receive an "echo" (previous-sender equals identifier), they mark themselves critical (big filled circle) and the others mark themselves non-critical. Subsequent packets are forwarded only by nodes A and C irrespective of originator (source independence). 47

5.4 Simulation results for the increasing network size scenario where the size ranges from 10 to 100 nodes but the density remains constant. 56

5.5 Simulation results for the increasing density scenario where the network size is 100 nodes but density ranges from 2.77 to 25 nodes/km^2 (i.e., simulation area ranges from 36 to 4 km^2). 58

5.6 Simulation results for the increasing network load scenario where the network size is 30 nodes but the data packet interval ranges from 5 to 0.5 seconds. 59

5.7 Simulation results for the increasing node speed scenario where the network size is 100 nodes but the node speed ranges from 4 to 20 m/s. 60

5.8 Simulation results for the increasing network size scenario where the size ranges from 10 to 100 nodes, the density remains constant, and the data rate is 1 Mbps. 61

6.1 An example VINE operation, in which node A sends a packet to node I. Several nodes learn gradients towards the source, the sender, and the previous sender of the packet as well its E2E-A. Shaded boxes indicate new states learned, and unshaded boxes indicate states carried over from the previous step. 73

6.2 Average communication complexity 76

6.3 Churn analysis showing the "sweet spot" between VINE and Flooding. 77

6.4 Simulation results for the increasing network size scenario where the size ranges from 10 to 50 nodes but the density remains constant. . 79

6.5 Simulation results for the increasing density scenario where the network size is 30 nodes but density ranges from 0.83 to 7.5 nodes/km^2 (i.e., simulation area ranges from 36 to 4 km^2). 80

6.6 Simulation results for the increasing network load scenario where the network size is 30 nodes but packet interval ranges from 10 to 30 secs. 81

6.7 Simulation results for the increasing network size where nodes are configured to transmit at 1 Mbps data rate. 82

7.1 A network with SDNC and one possible set of critical nodes (shaded). 87

7.2 All nodes learn routes to node J. The circles around nodes represent broadcast transmissions, while the arrows represent unicast. 94

7.3 Comparison of average communication complexities from an ns3 simulation experiment and Equation 7.5 using different link break probabilities. 97

7.4 Simulation results for scenario 1 (Increasing Network Size) where network size ranges from 60 to 100 nodes but density remains constant. 99

7.5 Simulation results for scenario 2 (Decreasing Network Density) where network size is 100 nodes but density ranges from approx. 11 to 2 nodes/km^2 (simulation area between 9 km^2 and 49 km^2). 101

7.6 Simulation results for scenario 3 (Increasing Network Load) where network size is 100 but data packet interval ranges from 10 to 2 seconds. 103

7.7 Simulation results for scenario 4 (Increasing Node Speed) where network size is 100 but node speed ranges from 2 to 10 m/s. 104

8.1 Simulation results for scenario 1 (Increasing Network Size) where network size ranges from 60 to 100 nodes but density remains constant. 112

8.2 Simulation results for scenario 2 (Decreasing Network Density) where network size is 100 nodes but density ranges from approximately 11 to 2 nodes/km^2 (i.e., simulation area from 9 to 49 km^2) 113

8.3 Simulation results for scenario 3 (Increasing Network Load) where network size is 100 but data packet interval ranges from 10 to 2 seconds. 114

8.4 Simulation results for scenario 4 (Increasing Node Speed) where network size is 100 but node speed ranges from 2 to 10 m/s. 115

8.5 Simulation results for networks of sizes up to 250 nodes. 116

9.1 (a) SDNC initiated TD flooding with cluster diameter (K) = 3. (b) Nodes learned their Route to SDNC (RTS). Critical nodes are shown shaded. 120

9.2 (a) TD message with K=0 resulting in nodes becoming candidates for Cluster Head. (b) Candidate CH node 24 randomly becomes first to initiate its TD flooding with K = 3. 121

9.3 Cluster formation at the end of TD flooding. 122

9.4 Nodes 3, 8, 10, 13, 18, 20, 22, and 27 becoming gateway nodes at the end of TD flooding. 123

9.5 Hierarchical view of the clusters formation. SDNC and CHs sending RU messages as network-wide-broadcasts. 124

9.6 CH node 28 sending CI message and gateway nodes forwarding it. . 125

9.7 Simulation results for scenario 1 (Increasing Network Size) where the network size ranges from 100 to 250 nodes but the density remains constant. 132

9.8 Simulation results for scenario 2 (Decreasing Network Density) where network size is 100 nodes but density ranges from approximately 5.5 to 2.5 nodes/km^2 (i.e., simulation area from 36 to 81 km^2) 133

9.9 Simulation results for scenario 3 (Increasing Network Load) where network size is 200 but data packet interval ranges from 10 to 2 seconds. 134

9.10 Simulation results for scenario 4 (Increasing Node Speed) where network size is 200 but node speed ranges from 2 to 10 m/s. 135

A.1 Packet Format 157

A.2 Packet Header Format 158

A.3 Message Format 158

A.4 Message Header Format 159

A.5 Message Body Format 160

A.6 Address Block Format 161

A.7 TLV Block Format 163

A.8 TLV Format 164

A.9 Packet Header 166

A.10 Message Format 167

A.11 TD Message Header 167

A.12 TD Message TLV Block 167

A.13 NI Message Header 168

A.14 NI Message TLV Block 168

A.15 NI Address Block 169

A.16 RR Message Header 169

A.17 RR Message TLV Block 169

A.18 RR Address Block 170

A.19 RU Message Header 170

A.20 RU Message Block TLV 170

A.21 Each route sent individually. 171

A.22 Destination IP addresses with the same next hop IP address are included in the same address block. 172

A.23 All destination IP addresses are include in the same address block. 172

A.24 Routes to forward packets based on multiple fields. 173

A.25 RUA Message Header 174

A.26 RUA Message TLV Block 174

A.27 CI Message Header 175

A.28 CI Message TLV Block 175

A.29 CI Address Block And Address Block TLV Block Pairs 175

ABSTRACT

A mobile ad hoc network (MANET) is a self-organizing infrastructure-less network of mobile nodes needing multi-hop communication. MANETs support a wide range of applications in vehicular, mesh, sensor, and IoT networks, as well as in military operations, emergency search and rescue operations, disaster relief efforts, and providing Internet connectivity to remote regions. MANET characteristics make routing packets challenging because node mobility results in dynamic changes to the network topology. Unsynchronized transmissions result in increased interference, packet losses, and link instability, making communication unreliable, especially over multi-hop routes.

Traditional MANET solutions have followed the decentralized paradigm, in which the nodes select routes for forwarding data packets either independently or as a group. In the past few years, Software-Defined Networking (SDN) has introduced a paradigm shift, in which centralized architectures have been sought-after for both wired and wireless networks. These architectures are designed to have an SDN Controller (SDNC), responsible for selecting network routes and dynamically controlling the network behavior in a centralized manner. Most existing SDN-based architectures for ad hoc networks propose using one or more of the following: infrastructure for hosting fixed SDNC, out-of-band single-hop links for control communication, location-tracking for learning network topology, or pre-existing IP connectivity for control communication. Such architectures are inadequate for infrastructure-less networks having intermittent links and susceptibility to high interference, packet losses, and collisions.

In this dissertation, we present an architecture for Software-Defined Mobile Ad Hoc Network (SD-MANET). We presume none of the constraints imposed by the previous architectures. We design our SD-MANET architecture to have the following three

functions: (1) learning route to SDNC, (2) learning network topology, and (3) sending network routes. We cater to the needs for proactive, reactive, and hierarchical routing protocols for MANETs by designing the following SD-MANET routing protocols:

1. PCC, a proactive routing protocol
2. CORR, a reactive routing protocol
3. CPR, an improved proactive routing protocol
4. HCPR, a hierarchical routing protocol

MANET applications using low data rates for providing long-range communication endure low network capacity. In several contexts, the capacity is so low that the use of control packets completely overwhelms the network, rendering all existing solutions inadequate even for moderate-size networks. We address this issue by designing two novel zero-control-packet protocols:

1. ECHO, a protocol designed for efficient network-wide broadcast
2. VINE, a protocol for delivering a message to the specified destination

These protocols do not use any control packets whatsoever. Instead, they include some additional information in the data packet header for building states in the nodes. Subsequent data packets get forwarded based on these states. The additional information in the data packet header is constant in size and does not scale with network size or density. We show it to be insignificant compared to the control packet sizes used by traditional routing solutions. We use some of the features of the ECHO and VINE protocols for designing the CORR, CPR, and HCPR protocols.

We evaluate all our routing protocols for a wide range of scenarios, addressing scalability, load, density, and mobility. We present their simulation results and theoretical analysis. The results indicate that our SD-MANET protocols perform as good as, and in most scenarios outperform, the state-of-the-art protocols. In low-capacity networks, where all existing solutions perform unsatisfactorily, our protocols not only meet the requirements but also scale with network size and show superior reliability.

Chapter 1

INTRODUCTION

Advances in wireless technologies have allowed rapid development of independent mobile networks. One popular category of mobile networks is Mobile Ad Hoc Networks (MANETs), in which nodes autonomously self-organize and establish wireless multi-hop communication among themselves. There are no dedicated forwarding devices in such a network. Each node is both a forwarding device and an end host. MANETs have a wide range of applications in several scenarios, including emergency search and rescue operations, military operations, disaster relief efforts, Internet connectivity, vehicular networks, and sensor networks. Based on their applications, they usually get classified into the following categories:

- **Tactical MANET [114]:** Military operations need on-the-fly wireless multi-hop connectivity in remote locations where infrastructure such as base stations do not preexist.
- **Vehicular Ad hoc Network (VANET) [54]:** Autonomous and self-driving vehicles need to communicate among themselves and with roadside units for collecting information for better decision-making.
- **Wireless Mesh Network (WMN) [23]:** Devices such as goTenna Mesh [3] provide off-grid communication during disaster relief and emergency situations using wireless peer-to-peer connectivity typically over multiple links.
- **Wireless Sensor Network (WSN) [22]:** Sensors collecting environmental data such as noise, temperature, humidity, and pressure need to transfer the collected information to the sink node.

Most MANET applications require establishing local communication within a group of nodes, but they may also be used to extend fixed infrastructure deployments. One such example is extending wired networks and providing Internet connectivity

over multi-hop access points. Depending on the application, the nodes may be mobile or stationary. They may also be a part of the existing infrastructure or deployed independently. This dissertation considers MANETs to be infrastructure-less having mobile nodes that need communication among themselves. Designing solutions for such networks is challenging due to the following reasons:

1. Node mobility results in a dynamic and unstructured network topology.
2. Nodes have relatively unstable links, mainly due to mobility, interference, and propagation losses from changes in the environmental conditions.
3. Link instability results in unreliable communication, especially over multi-hop links.

There are several aspects of designing solutions for such networks. The two most important ones are (1) utilizing the shared wireless medium efficiently for transmitting messages and (2) sending messages efficiently from source to destination. This dissertation focuses on the second aspect. In particular, it presents protocols for delivering packets from one node to another (i.e., routing protocols).

Traditional MANET solutions have followed the decentralized paradigm, in which the nodes select their routes for forwarding packets either independently or as a group. Considering the popularity of MANETs and their wide range of applications, researchers have proposed several routing protocols in the literature [31]. Most of them are adaptations of distance-vector or link-state routing and are generally classified into three categories: *proactive*, *reactive*, and *hybrid*.

1.1 Software-Defined Networking

In the past decade, the advances in Software-Defined Networking (SDN) have brought about a paradigm shift, in which centralized architectures have been sought-after for both wired and wireless networks. The main principle of SDN is to separate functions, such as routing, from the forwarding devices and move them to a logically centralized entity, called SDN Controller (SDNC). This architecture allows the SDNC

to manage the forwarding devices in a centralized manner. It also enables provisioning of additional services, such as load balancing, firewall, access control, QoS, and network monitoring, without the need for dedicated devices for each of them. These features make the network programmable and improve the management and utilization of the available network resources. The control communication between the SDNC and the forwarding devices is through a well-defined protocol such as OpenFlow [84] or ForCES [43].

The SDN design principles apply to both wired and wireless networks. A large number of wireless domains have adopted and benefited from the centralized architectures [60, 59]. However, the application of SDN to MANETs faces several challenges, mainly due to the dynamic nature of network topology. The traditional SDN architecture is not designed to accommodate the requirements of a DIL (Disconnected, Interrupted, and Low-bandwidth) network. It requires reliable control communication between the SDNC and forwarding devices. The existing control communication protocols also do not have the functionalities necessary for managing MANETs. The sizes of control messages (i.e., OpenFlow messages) may also be too large for a scarce bandwidth MANET.

Despite the challenges, a centralized architecture for MANETs can potentially realize the following benefits:

1. Moving the complex topology discovery and route selection procedures from nodes to the SDNC increases the battery lifespan of nodes.
2. Avoiding the need for regularly sharing routing information between all neighbor nodes reduces the communication overhead.
3. Dynamically adjusting the routing parameters based on the network characteristics may improve the performance.
4. Opportunistically updating the routes in all nodes on receiving route request or identifying topology changes reduces the latency.
5. Preemptively sending routing information if node movement patterns are available in advance enables uninterrupted and seamless communication.

6. Provisioning nodes with multiple radios configured to work on different frequencies (or RF technologies) may allow the SDNC to utilize the radio spectrum efficiently and improve the network performance.

There have been several SDN-based centralized architectures [38, 122, 41, 74, 109, 127, 61, 37, 29, 24] proposed for MANETs in the past few years. However, most, if not all, of them rely on one or more of the following constraints:

1. OpenFlow [84] or ForCES [43] for the control communication.
2. Network infrastructure (i.e., a base station) for hosting SDNC.
3. Single hop (direct) wireless link between the SDNC and each node, using either an in-band or out-of-band connection, such as cellular or LTE.
4. Location tracking services for learning the network topology.
5. Preexisting IP connectivity.

These constraints make the existing architectures inadequate for infrastructure-less MANETs having low-capacity links and susceptibility to high interference, collisions, and packet losses. Moreover, these architectures fail to address the network dynamics and autonomous network topology discovery [59, 40]. Further, they present use cases and improvements in certain situations but do not provide extensive evaluations and performance comparisons to the state-of-the-art solutions.

1.2 Dissertation Contributions

We propose an architecture for Software-Defined Mobile Ad Hoc Networks (SD-MANET) and design several centralized routing protocols for supporting different needs of MANETs. We also present two novel zero-control-packet routing protocols for the low-rate long-range MANETs.

Software-Defined Mobile Ad Hoc Networks Architecture

Our SD-MANET architecture is designed to have none of the constraints of the previous work on SDN-based MANET architectures. In particular, the SDNC is one of the mobile nodes within the network. All nodes, including the SDNC, have limited

transmission ranges and need multi-hop control and data communications. The SDNC learns the network topology without using any location service and selects routes for all other nodes in the network. It uses the in-band channel for communicating with the nodes. The SDNC also hosts data applications and relays data packets similar to other nodes.

We identify three functions to be necessary for managing an SD-MANET. They are (1) learning route to SDNC, (2) learning network topology, and (3) sending network routes. We use these three SD-MANET functions for describing a centralized proactive routing protocol called PCC.

Low-Rate Long-Range Routing Protocols

Several application contexts, such as public safety and disaster relief, require off-grid multi-hop communication for covering a large area. Keeping the network connected using short-range communication technologies (e.g., WiFi) needs dense deployments, which increases the cost. Thus, users need long-range devices that are preferably lightweight (and hence low power) and inexpensive. However, achieving long ranges requires using reduced data rates, assuming power and cost requirements remain the same. There are fundamental constraints on optimizing range, rate, cost, and power, all simultaneously.

In such low-rate regimes, the overhead of control packets of the routing protocol becomes intolerable because it consumes most of the available bandwidth. Further, at low rates, the transmission delay is high, and the network experiences increased interferences and packet collisions.

For addressing these challenges, we design two novel zero-control-packet routing protocols: ECHO and VINE. The ECHO protocol performs efficient network-wide broadcasts by selecting a subset of nodes in the network (i.e., the critical nodes), whose transmissions result in all nodes receiving the message. The VINE protocol delivers the message reliably to the destination specified in the header. Both these protocols do no use any control packets whatsoever. Instead, they use additional fields in the

packet header for building states, which the nodes use for forwarding subsequent data packets.

SD-MANET Routing Protocols

We use some of the features of the ECHO and VINE protocols for optimizing the three SD-MANET functions. These optimizations reduce communication overhead and increase reliability. We design reactive, proactive, and hybrid routing protocols that make use of these optimizations for efficiently learning the network topology and disseminating the routing information.

In our reactive protocol, called CORR, the SDNC sends routing information on receiving request messages from the nodes. By contrast, in our proactive protocol, called CPR, the SDNC sends the routing information periodically for maintaining up-to-date states in all nodes.

For addressing the scalability challenges, our hierarchical protocol, called HCPR, builds clusters of nodes in the network and identifies gateway nodes. Each cluster has a cluster head for configuring intra-cluster routing, whereas the gateway nodes enable inter-cluster routing.

Evaluations

Our SD-MANET architecture and protocols are implemented and evaluated in ns3 [13]. Since ns3 does not include a fully developed module of SDN, we extended the simulator to include the required features, making changes to several layers of the IP stack. These changes enabled us to integrate an SDN architecture into simulations of MANET networks. We use this enhanced ns-3 simulation framework to conduct detailed performance evaluation studies, comparing our protocols with other standard and widely-used MANET protocols.

We also present the communication complexity analysis of all protocols in their respective chapters. The control messages of all SD-MANET routing protocols are designed using the packet/message format specifications described in RFC 5444 [36].

We describe the designs of these control messages in Appendix A. The zero-control-packet routing protocols do not have any control messages, but their data packet header includes extra information, which we explain in the respective chapters.

1.3 Dissertation Outline

We show the chapter dependency and dissertation outline in Figure 1.1.

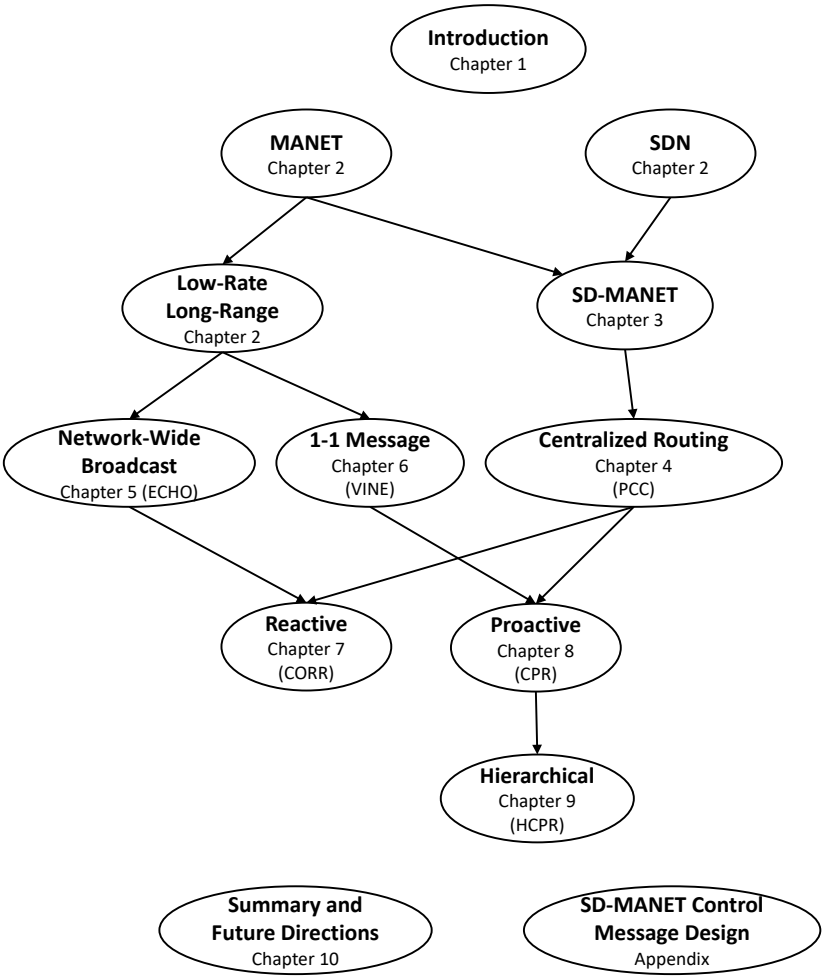


Figure 1.1: Dissertation Outline

Chapter 2 describes the background information on the traditional and SDN architectures followed by an overview of the MANET routing protocols proposed over

the years. It ends with overviews of the related works on SDN-based centralized architectures for MANETs and low-rate long-range networks.

Chapter 3 describes our SD-MANET architecture and the internal structure of an SD-MANET node. It also explains the modifications made to ns3 for evaluating our architecture and all the protocols. It ends with an overview of the opportunities facilitated by our SD-MANET architecture.

Chapter 4 describes our first proactive routing protocol, called Proactive Control Communication (PCC), for the SD-MANET architecture. It is our preliminary work that allowed us to gain insight into the inherent challenges of centralized routing and helped us design better approaches that address these challenges.

Chapter 5 describes our first zero-control-packet routing protocol, called ECHO, designed for network-wide broadcasts, such as collaborative mapping and emergency beaconing.

Chapter 6 describes our second zero-control-packet routing protocol, called VINE, designed for user-to-user communications. This protocol allows nodes to send private messages to other nodes as 1-1 messages.

Chapter 7 describes our reactive routing protocol, called Centralized Opportunistic Reactive Routing (CORR), designed for the SD-MANET architecture. This protocol addresses the challenges in centralized routing and presents improvements over the state-of-the-art routing protocol.

Chapter 8 describes our second proactive protocol, called Centralized Proactive Routing (CPR), designed for the SD-MANET architecture. It presents improvements over the previous proactive protocol (i.e., PCC) and also over the state-of-the-art protocol.

We use some of the features of the zero-control-packet protocols (ECHO and VINE) for designing the CORR and CPR protocols.

Chapter 9 describes our hierarchical routing protocol, called Hierarchical Centralized Proactive Routing (HCPR), designed for the SD-MANET architecture. This protocol allows the SDNC to address the scalability challenges.

Chapter 10 presents a summary of this dissertation and highlights our contributions. It also presents future research directions and lists some of our other research projects that have led to publications.

The Appendix first summarizes the RFC 5444 specifications and then describes the designs of the control messages used by our SD-MANET routing protocols.

Chapter 2

BACKGROUND AND RELATED WORKS

In this chapter, we present an overview of the traditional and Software-defined networking architectures and highlight the differences between them. Then we discuss the challenges associated with routing data packets in mobile ad hoc networks and summarize the protocols proposed over the years for addressing these challenges. Later, we characterize the low-power long-range networking and describe their applications and existing technologies. In the end, we review the centralized SDN-based architectures proposed for mobile ad hoc networks over the years.

2.1 Traditional Network Architecture

Network architectures can be explained using two components: the data plane and the control plane. The main function of the data plane is *forwarding*, i.e., the process of forwarding a received packet to the next hop. Typically, all traditional architectures use the destination-based forwarding model, i.e., forwarding packets based on the destination IP address in the packet header. The main function of the control plane is *routing*, i.e., determining the end-to-end route for forwarding packets from the source to the destination. Figure 2.1 represents a traditional network architecture, in which the control and data plane functions are bundled together in each router.

A key component in the data plane is the forwarding table. It has entries for IP addresses for matching against the incoming packets and determining the actions to be taken. The forwarding function matches the destination IP address in the packet header against the entries in the forwarding table. The routing function configures these entries in the forwarding table using the routing protocol.

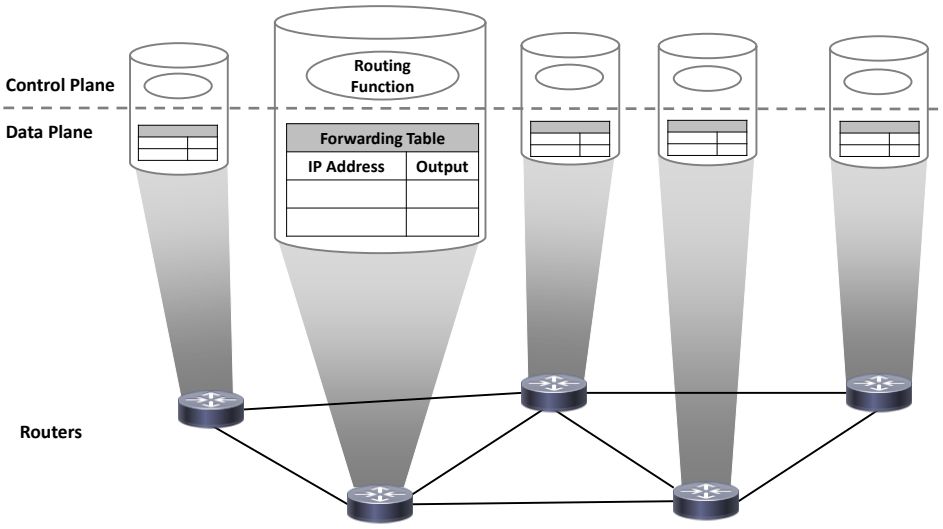


Figure 2.1: Traditional Network Architecture

A general classification characterizes the routing protocols into two categories: distance vector and link-state. Most link-state routing protocols select routes in a centralized manner, while most distance-vector routing protocols select routes in a distributed manner.

In a link-state routing protocol, each router first collects the status of all its links to the neighbor nodes (i.e., the link-state) and disseminates that information in the network, so that each router has the same global state. Then, it uses a centralized algorithm for selecting least-cost routes to every destination. Cost is typically the number of hops to the destination but could also be a function of other link metrics. Most protocols use the Dijkstra's shortest path algorithm [42] for selecting the routes.

In a distance-vector routing protocol, the selection of routes is carried out in an iterative and distributed manner. No single router has a complete view of the network. Instead, each router begins with only the knowledge of the costs of the directly connected links. Then, through an iterative process of route selection and exchange of information with other neighbors, routes to all other routers are selected. Irrespective of the approach, the goal of every protocol is to select efficient routes for

forwarding data packets in the network.

2.2 Software-Defined Networking Architecture

The Software-defined networking architecture separates the control plane functions from routers and moves them to a logically centralized entity called the SDN Controller (SDNC). The forwarding function remains in the router, but the SDNC is responsible for performing the routing function. Because of this separation, this architecture refers to the routers as the forwarding devices. Figure 2.2 shows the SDN architecture and the separation of control and data planes.

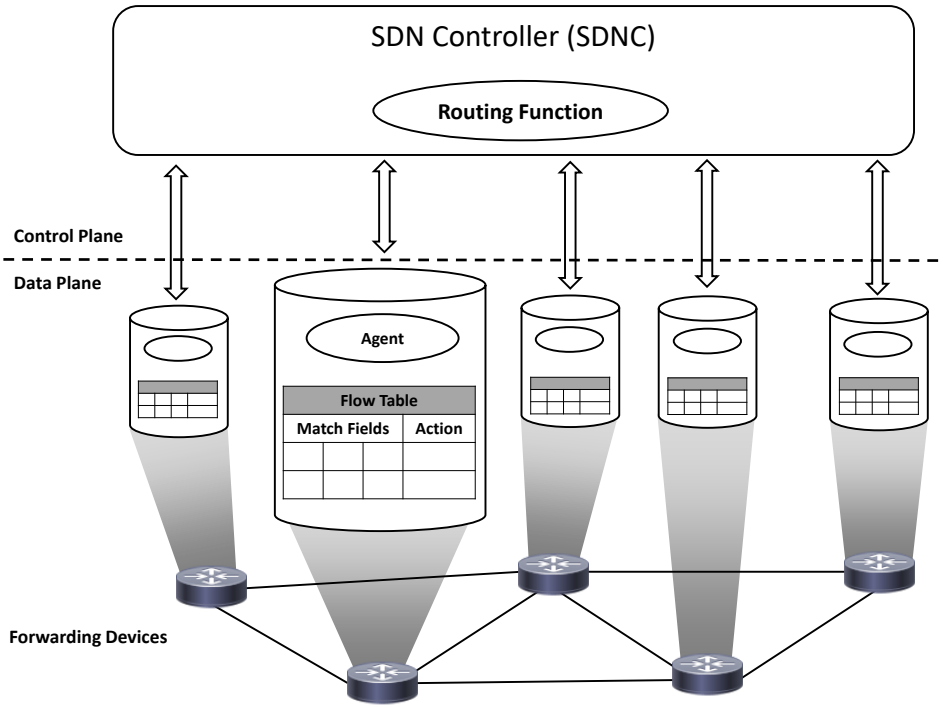


Figure 2.2: Software-Defined Networking Architecture

The SDNC collects the global network state and selects the routes in a centralized manner. Each forwarding device has an agent for communicating with the

SDNC. This communication is through a well-defined protocol – most common being the OpenFlow protocol [84]. Typically, these agents do not communicate among themselves, nor do they actively take part in the route selection process.

The forwarding function used by this architecture is more generalized than the one used by the traditional architecture, i.e., packets are forwarded not just based on their destination IP address but a combination of several fields in the header. Consequently, the structure of the forwarding table is also different. It is characterized as a "match-plus-action" table and called the "flow table".

In contrast to the traditional forwarding table, the flow table can match several fields in the headers of the incoming packets against entries. Figure 2.3 shows a complete list of these fields; among them, eleven are in the packet header, and one is the ingress port. These specifications are from OpenFlow specification 1.0 [14].

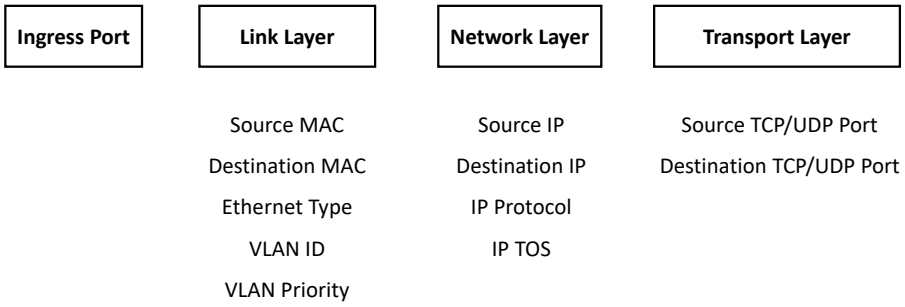


Figure 2.3: Packet matching fields in flow table (OpenFlow 1.0)

The flow-based model allows forwarding packets based on all twelve fields shown in Figure 2.3. Unmatched packets can be either dropped or sent to the SDNC for further processing. The flow table also maintains a set of counters for each entry and updates it for every hit. The SDNC can collect these flow statistics and identify traffic patterns for making intelligent routing decisions. Each entry has a set of instructions associated with it for determining the actions to be taken on matched packets – e.g., forward the packet, drop the packet, and rewrite header fields. Each entry can have

a wide variety of actions. These actions can be updated easily by the SDNC for realizing network functions such as load-balancing, QoS, firewall, and security, making the packet forwarding, and hence the network, programmable.

To summarize, the two key characteristics of SDN architecture are (1) separation of data plane and control plane and (2) flow-based forwarding model.

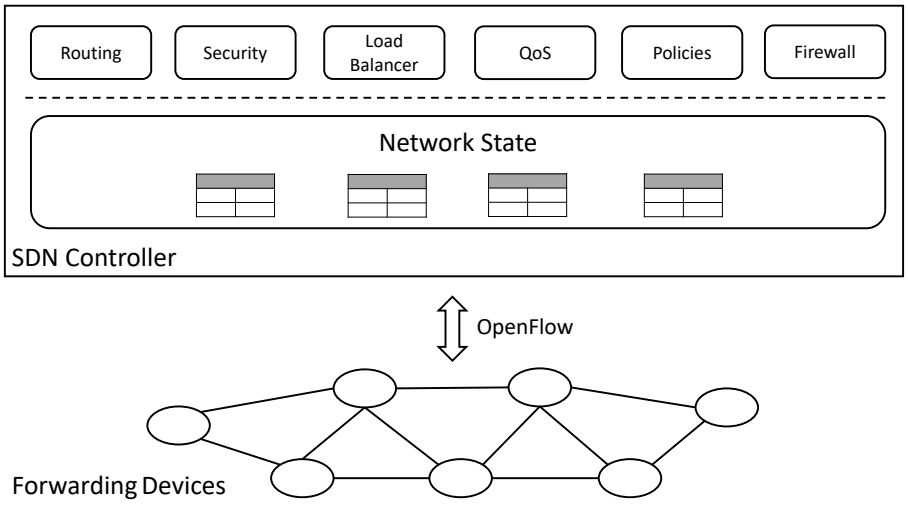


Figure 2.4: SDN Controller

Figure 2.4 shows the internal architecture of the SDNC. The routing function, along with other functions, such as load balancer, QoS, policies, and firewall, runs inside the SDNC as an application. The SDNC maintains the network state, such as network topology, link costs, and flow statistics. The flow-based forwarding model allows the entries (i.e., flow rules) to have priorities for realizing QoS and providing differential services to the network traffic. The QoS applications can determine these priorities. The load balancer application can balance the network traffic according to the workload. The flow-based forwarding model also allows the installation of firewall rules that can block packets based on the twelve fields shown in Figure 2.3. Firewall applications can track TCP/UDP connections and set entries for restricting untrustworthy connections. Security applications can enable Intrusion Detection System by

matching packets against the signature of known attacks. Applications can also be added to implement policies provided by the users as well as providers.

These applications utilize the network state information and their requirements for configuring entries in the flow tables. A protocol, such as OpenFlow, determines the communication between them. The ability to control and manage the entire network via these applications makes the network programmable.

2.3 Mobile Ad hoc Networks

A Mobile Ad hoc Network (MANET) consists of a group of mobile nodes that communicate without requiring a fixed wireless infrastructure. There are no dedicated forwarding devices in the network, and each node acts both as a forwarding device and an end host.

Some of the unique characteristics of MANETs are (1) node mobility causing dynamic changes in the network topology, resulting in intermittent links, (2) nodes having limited wireless transmission range and needing multi-hop communication, (3) unsynchronized transmissions leading to interference and collisions, and (4) propagation losses causing link instability.

From the routing perspective, these characteristics are so different than those of wired (and other wireless) networks, that it resulted in designing a new class of routing protocols called the MANET routing protocols. There have been several MANET routing protocols proposed in the literature. Most of them are an adaptation of distance-vector or link-state routing.

In a distance-vector routing protocol, nodes advertise their distance to each node as a vector of distances. Nodes receive such advertisements from each of their neighbors and combine them for selecting their best route to each destination. In a link-state routing protocol, nodes periodically monitor their links to neighbor nodes using Hello messages and then broadcast the information to all other nodes. Each node builds and maintains up-to-date topology information and uses it for individually selecting routes to every other node. Both distance-vector and link-state protocols can

be designed to be proactive or reactive. A generic classification categorizes them into proactive, reactive protocols, and hybrid [31].

MANETs with infrequent network traffic typically use reactive routing protocols. Reactive represents a subclass of MANET routing protocols characterized by their on-demand selection of routes. A few examples are Dynamic Source Routing (DSR) [70] and Ad hoc On-demand Distance Vector (AODV) [92]. Typically, execution of a route request procedure – involving flooding request messages – results in the source learning a route to the destination. Protocols may also use a route maintenance procedure for maintaining the previously learned routes. These protocols may experience a high delay caused by data packet buffering while executing the route request procedure.

On the other hand, MANETs with frequent and regular network traffic use proactive routing protocols, in which routes are selected in advance between all pairs of nodes in the network. A few examples are Destination-Sequenced Distance-Vector (DSDV) [91] and Optimized Link State Routing (OLSR) [69]. Selecting routes in advance and proactively updating the routing tables of all nodes incur high routing overhead for attaining the desired network throughput and delay caused by periodic dissemination of routing information throughout the network. Different protocols disseminate different types of routing information, depending on their link-state or distance vector characteristics.

Hybrid routing protocols combine the elements of both proactive and reactive. A few examples are Zone Routing Protocol (ZRP) [103] and Fisheye State Routing (FSR) [88]. The general idea is to characterize zones and enable proactive routing within the zone and reactive routing between the zones. These protocols aim for a trade-off between the overhead and delay based on the network characteristics, such as size, density, and mobility. However, realizing an optimal trade-off is difficult when network characteristics are not available or vary with time.

A few routing protocols [72, 71] use location services, such as GPS, for tracking the positions of nodes and selecting routes between them. However, these protocols

face difficulties when the actual topologies based on path loss and connectivity are different from the estimated ones.

Scalability is another challenge in MANET routing. There could be several ways to define scalability. The most general definition is the ability of the routing protocol to perform efficiently with the growth of one or more network parameters, such as network size, network density, node speed, and traffic load [53]. The protocols mentioned so far experience deterioration in performance when any of these parameters grow. Several hierarchical routing protocols [67, 106] address these scalability issues. These protocols build a hierarchy of nodes, typically through clustering techniques. Nodes at the higher levels of hierarchy provide additional services like acting as gateways and forwarding the packets to other clusters.

Even though several classes of protocols operate under different scenarios, they usually share common goals, such as reducing overhead, maximizing the throughput, and minimizing the delay. The differentiating factor between the protocols is the way they select and maintain the routes. Almost all protocols have their deployment scenarios and parameter combinations where they outperform other protocols. Optimal selection often requires careful analysis of the network scenario, requirements, and characteristics.

2.4 SDN-based Architectures for MANETs

Most SDN-based centralized architectures [38, 122, 41, 74, 109, 127, 61, 37, 29, 24] proposed for MANETs expect an infrastructure-based deployment, where the SDNC is stationary and uses either single hop (direct) out-of-band (i.e., a different network) links or preexisting in-band (i.e., the same network) IP connectivity for control communication with the nodes (sensors [38], mesh [41], vehicles [74, 109, 127, 61]). Some [61, 38] use a location-tracking service for learning the network topology and enabling multi-hop data communication, while others [41, 29] present QoS and traffic engineering use cases. However, these architectures are inadequate for infrastructure-less

networks. Further, the use of OpenFlow or any similar protocol would be impractical for low-capacity networks because of their large overhead.

An SDN-based architecture for mesh networks [40] suggests using OpenFlow-enabled mesh access points and gateways, allowing the SDNC to trigger and manage the handover of mobile nodes between access points. A traffic engineering use case in [41] dynamically configures rules for balancing the Internet traffic across multiple gateways. However, in both these approaches, the multi-hop routing in the network is configured using OLSR. An SDN-based architecture for sensor networks proposed in [38] deploys the SDNC in a stationary base station and assumes that a direct link is available to each sensor node for the control communication. The SDNC learns the network topology by tracking the position of nodes using a location service and then uses it for configuring the network routes.

SDN-based architectures [75, 73, 109] for vehicular networks also propose deploying the SDNC inside a base station. Architectures in [127, 61] assume vehicles to remain connected to the Internet or cloud infrastructure via an out-of-band cellular network. For issues related to loss of connectivity with the SDNC, a clustering technique in [37] creates domains, each having a cluster head acting as the local SDNC. Vehicles periodically share their position, velocity, direction vectors, and cluster information. A local SDNC receives route request messages from vehicles in its cluster and forwards the messages to other SDNCs via a gateway selection algorithm for finding a path to the destination. All these architectures require infrastructure for hosting the SDNC and out-of-band links, in the form of LTE or WiFi/ WiMax, for the control communication.

A hierarchical distributed control plane architecture proposed in [44] divides the network into domains. Each domain has its own SDNC. The routing functions remain distributed, while the SDNC performs all other control functions.

In contrast to all the above approaches, we have designed an architecture [45] (explained in Chapter 3), in which one of the mobile nodes hosts the SDNC. All nodes, including the SDNC, have limited ranges, and the architecture uses multi-hop

in-band control and data communications. The SDNC learns the network topology and disseminates the network routes without tracking the locations of mobile nodes.

For our centralized architecture, we have designed several different centralized routing protocols [45, 49, 46, 85], catering to the needs of proactive, reactive, and hierarchical routing approaches. We describe these protocols in Chapters 4, 7, 8, and 9.

2.5 Low-Rate Long-Range Networking

The essential requirement for wireless communication is that there needs to be connectivity, i.e., the node should have a link to another node. Consequently, if the node wants to be able to communicate with every other node, then the graph representing the network topology should be connected. Connectivity depends on wireless transmission range, which in turn depends on the RF technology being used. Each technology provides a different trade-off between range, data rate, cost, and power [95]. But for given output power, the data rate determines the range. A low data rate provides a long transmission range due to increased sensitivity at the receiver. Further, long messages sent at low data rates increase the possibilities of interference. So, long-range systems typically need to optimize the range and transmission time balance. Narrowband techniques (25 kHz bandwidth) provide an excellent link budget due to low in-band receive noise and give an optimum trade-off between range and the transmission time. The industry widely accepts and utilizes these technologies for long-range systems [76].

The proliferation of the Internet of Things (IoT) applications – needing long-range and wide-area communication at low-power – have allowed the development of Low-Power Wide-Area Networks (LPWAN), which supports the development of long-range, low-power, and low-cost devices as well as infrastructure for connecting a large number of devices [66]. The increasing demands have resulted in the development of several competing LPWAN technologies. These technologies combine low data rate and robust modulation schemes to achieve multi-kilometer communication ranges. Some of the infrastructure-less LPWAN technologies are LoRa [10], SigFox [15], IQRF [8],

RPMA (Ingenu) [7], DASH7 [1], Weightless-N (nWave) [16], Weightless-P [16], SNOW (Sensor Network Over White Spaces) [101, 102], while some of the infrastructure-based technologies are LTE Cat M1 [11], EC-GSM-IoT [2], NB-IoT [115], and 5G [12].

LoRaWAN [10] is one of the successful ones, having its network stack rooted in the LoRa physical layer, which uses data rates between 0.3 Kbps and 50 Kbps. LPWAN provides opportunities in a large class of IoT applications: smart city [123], Agriculture and Smart Farming [111], Healthcare Applications [65], and Transportation [119].

Despite several promises, existing LPWAN technologies face several hurdles because of spectrum limitation, coexistence, mobility, scalability, coverage, security, and application-specific requirements, such as traffic and real-time communication, making their application challenging [66].

Among all the above challenges, scalability in dense networks is the biggest challenge for LPWANs [39]. The performance of LoRa, widely considered as an LPWAN leader [5, 9, 28, 34, 83], drops exponentially as the number of end-devices grows [21, 25, 30, 39, 57, 112]. Further, most applications are limited to star topologies, whereas the cellular-based ones rely on wired infrastructure for integrating multiple networks and covering larger areas. Lack of proper infrastructure and connectivity limit their agricultural applications.

Zigbee [17] is an IEEE 802.15.4-based specification [6] for a suite of high-level communication protocols used for creating small low-power digital radios. It has a defined rate of 250 kbps best suited for intermittent data transmissions from a sensor or input device and applications, such as for home automation, medical device data collection, and other low-power low-bandwidth scenarios. Zigbee devices transmit messages over long distances using a multi-hop mesh network of devices.

Narrowband tactical communications (e.g., NATO NBWF [18]), off-grid disaster relief, and public safety professionals often need an instantly and inexpensively deployable off-grid communications system for collaborative mapping, texting, and emergency beaconing [96]. They need communication devices to be lightweight, long-lasting (low

power), and not expensive (low cost). Besides, they need a multi-hop off-grid connectivity for covering a large area (i.e., mesh networking) as well as support user mobility. Because of all these requirements, the devices are required to use low data rates.

In all the above contexts, using low data rates is not a problem in itself, but it is essentially a routing protocol problem. The routing protocols use different strategies for generating and disseminating control packets, and some may do the job with fewer control packets. However, it turns out that the very act of using control packets consumes a base level of overhead. In networks with high data rates, the overhead of these control packets is tolerable. However, when the data rate is low, the overhead consumes most of the available bandwidth, leaving little or none for the actual traffic.

For addressing the above issue, we have proposed a mobile mesh networking architecture in [98] for networks characterized by low data rates and designed two zero-control-packet routing protocols [47, 97, 48]. Our first zero-control-packet protocol, called ECHO [47, 97], is explained in Chapter 5. It allows nodes to perform efficient network-wide broadcasts by selecting and maintaining a broadcast backbone without using any control packets. Our second zero-control-packet protocol, called VINE [48], is explained in Chapter 6. It delivers the packet to the destination specified in the header. VINE includes some information in the packet header that allows nodes to build states (i.e., routes) towards nodes. Nodes use these states for forwarding subsequent data packets to their destinations.

We use some of the features of our zero-control-packet protocols (ECHO and VINE) for designing the centralized routing protocols for our SDN-based architecture for MANETs.

Chapter 3

SOFTWARE-DEFINED MOBILE AD HOC NETWORK

In this chapter, we first describe our architecture for Software-Defined Mobile Ad Hoc Networks (SD-MANETs) and follow it up with an explanation of the internal structure of an SD-MANET node. Then we illustrate the modifications made to the ns3 simulator for evaluations. In the end, we explain the opportunities enabled by an SDN-based centralized architecture for MANETs.

3.1 Design Considerations

In our design considerations for SD-MANETs, we *do not* assume the availability of any of the following: (1) infrastructure for hosting the SDNC, (2) single-hop (direct) out-of-band control communication links between the SDNC and each node, (3) location services for tracking the position of nodes or for learning the network topology, and (4) preexisting IP connectivity for control communication.

Using infrastructure such as an LTE base station for hosting, makes the SDNC immobile, but we consider all nodes, including the SDNC, to be mobile. Using single-hop control communication links requires the SDNC to have a better (longer) transmission range than other nodes, but we have considered all nodes to possess similar transmission capabilities.

3.2 Architecture

Figure 3.1 shows our SD-MANET architecture, where one of the mobile nodes hosts the SDNC. This node is predetermined and not based on any election procedure. It has functionalities similar to other nodes in the network, including hosting data applications and relaying data packets. There are no dedicated forwarding devices.

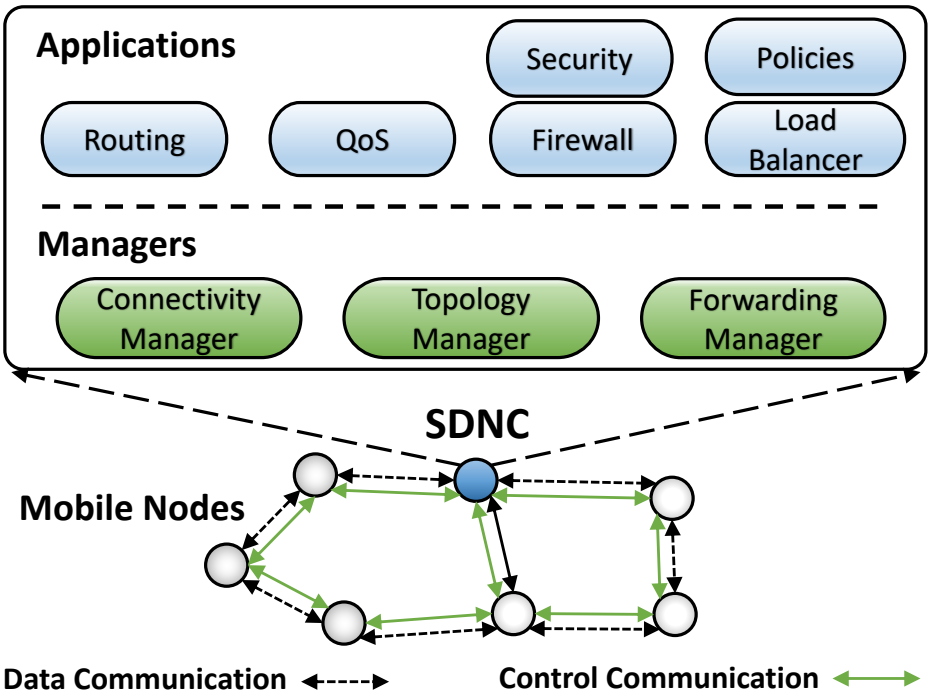


Figure 3.1: Software-Defined Mobile Ad hoc Network Architecture

Each node acts both as a forwarding device and an end host. Each node has limited wireless transmission range, so both the control and data communications are over multi-hop routes.

In the traditional MANET architecture, the data plane and control plane functions are bundled together in each node, and the routing functions inside the nodes communicate with each other for selecting routes individually. By contrast, in our architecture – which is similar to the standard SDN architecture – the routing function is moved from all nodes to the SDNC.

With the SDNC having the routing function, it has responsibility for selecting routes for all nodes in the network. The SDNC selects the routes using the network's global view, which it learns and maintains by periodically learning the network topology. One way of learning the network topology is by knowing the neighborhood information of each node. Each node needs a route to the SDNC for sending its

neighborhood information. Thus, our architecture requires the following three network functions: (1) learning route to SDNC, (2) learning network topology, and (3) sending network routes. The three managers inside SDNC (shown in Figure 3.1) are responsible for these three functions. We describe them below.

- **Connectivity Manager:** In a dynamic network, the network topology changes frequently, but nodes need to learn and maintain their route to SDNC for sending control messages. The Connectivity Manager helps nodes to perform this task.
- **Topology Manager:** The SDNC needs to learn the network topology for selecting the network routes. The Topology Manager is responsible for collecting the messages having the topology information.
- **Forwarding Manager:** The SDNC needs to send the route updates to all nodes. The Forwarding Manager is responsible for preparing the messages and disseminating them in the network.

The three managers help SDNC manage the network in a centralized manner and address the traditional MANET challenges: dynamic network topology, multi-hop communication, and infrastructure-less deployment.

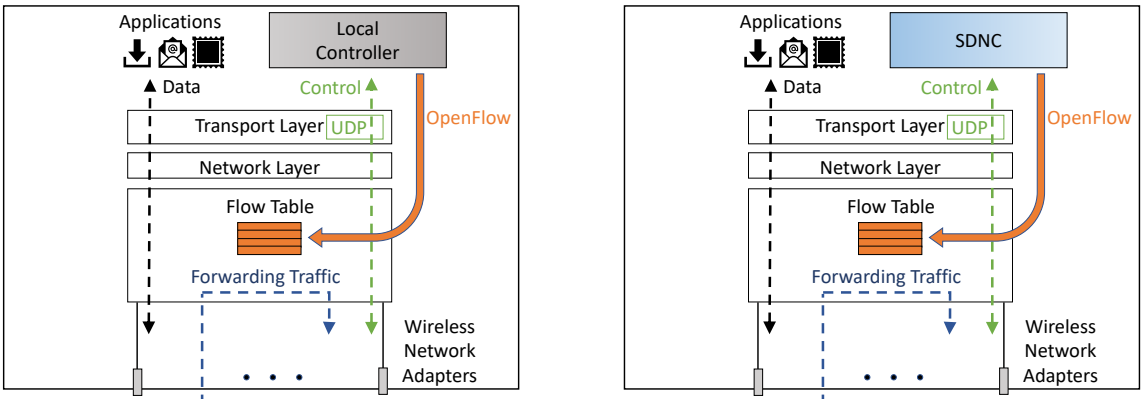
We have designed several different routing protocols supporting the SD-MANET architectures. All of them are described using the three functions listed previously.

In addition to the routing application, the SDNC also includes other applications, such as load balancer, QoS, and policies, that can determine and influence the route selection. The network becomes programmable through these applications.

3.3 Internal Structure of Node

Figure 3.2 shows the internal structure of an SD-MANET node. Each node has a flow table, which is similar to the flow table of an OpenFlow-enabled forwarding device. Data packets are forwarded using the entries (network routes) configured in the flow table[1].

[1] From here on, we use the terms flow table and routing table interchangeably.



(a) non-SDNC node (b) SDNC node

Figure 3.2: Internal structure of an SD-MANET node.

The routing function inside a traditional MANET node communicates with the routing functions in other nodes for updating the routing table by exchanging control messages typically as UDP payloads. In the standard SDN architecture, shown previously in Chapter 2, Figure 2.2, each forwarding device has an agent for communicating with the SDNC using the OpenFlow protocol and installing routes in the flow table. In our SD-MANET architecture as well, each node has an agent, called the local controller, which communicates with the SDNC using a communication protocol. The local controller maintains a route to SDNC for sending control messages. The communication between the local controller and the SDNC is not using OpenFlow but using our custom-designed protocols (explained in the following chapters). The local controller converts the routing information sent by the SDNC into the OpenFlow messages for installing routes in the flow table. This conversion reduces the communication overhead and makes the network programmable, allowing our architecture to facilitate several benefits of SDN.

Nodes can have more than one wireless adapters. But since the architecture does not use any out-of-band communication, nodes can use the same adapter for both control and data communications. For identifying the control packets, the flow table

has an entry for checking the port number in the transport header and sending the matching packets to the local controller.

3.4 ns3 Simulator Modifications

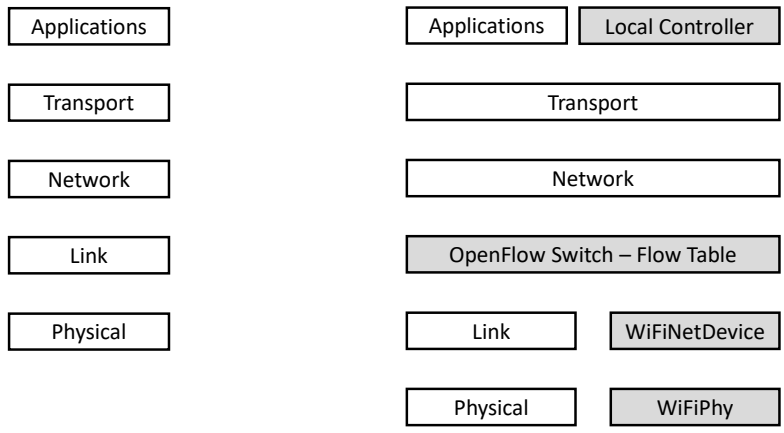
We now present the modifications made to the ns3 simulator for implementing our SD-MANET architecture.

The ns3 simulator has a rudimentary module available for SDN and OpenFlow simulations. This module includes components for an elementary controller and an OpenFlow-enabled switch supporting OpenFlow specification v0.8.9 (draft version) for simulating wired networks with Ethernet links between the host and the switch.

Figure 3.3(a) shows the traditional IP stack of a host in ns3. We modified this stack to include the OpenFlow switch component between the network layer and the link layer (shown in Figure 3.3(b)). Since the original switch component works only with Ethernet network devices, we modified it to work with WiFi network devices. We configured the devices to work in the ad hoc mode (i.e., infrastructure-less mode) and use CSMA.

We also modified the elementary controller to run as an application inside each node. The controller application uses UDP for communicating with the controller application in other nodes. We implement the routing protocol logic and control packet processing inside the controller application. We design the control messages used by the routing protocols using the specifications described in RFC 5444 [36].

Nodes use the entries in the flow table of the switch component for route lookup. We extended the OFSID library providing the flow table implementation for supporting the route lookup. When the data applications generate data packets, the node bypasses the network layer (i.e., routing table lookup) and sends the packets directly to the switch component. Similarly, when the node receives a packet from another node, it sends them to the switch (not the network layer) for matching against the flow table entries and determining the action.



(a) Traditional IP Stack (b) Modified IP Stack

Figure 3.3: Modifications to the IP stack in the ns3 simulator.

3.5 SD-MANET Opportunities

In addition to inheriting all the traditional advantages of SDN, our architecture also enables several opportunities for improving the performance of MANETs. These opportunities include:

Control communication overhead: Most decentralized routing protocols require periodic exchange of routing information between all pair of neighbor nodes. Disseminating the routing information from a centralized location (i.e., the SDNC) can reduce the communication overhead, which is one of the objectives of all routing protocols, especially in a scarce bandwidth MANET.

Dynamic adjustment of routing parameter values: Most routing protocols require pre-configured values for the rate at which the functions, such as neighbor discovery and route discovery, are performed. A centralized architecture allows estimating appropriate values for the routing parameters based on the global view of the network and then programmatically setting these values in the nodes via control messages.

Latency: Node mobility causes dynamic changes in the network topology and

frequent link failures. A centralized architecture can quickly identify the configured routes affected by the changes in the network topology and opportunistically update them for reducing the latency.

Route planning: In scenarios where the movement patterns of nodes are available in advance, the SDNC can preemptively update routes in all nodes and facilitate seamless communication.

Spectrum utilization: Nodes having multiple wireless adapters working at different frequencies can be used for forming subnets or virtual wireless networks. Nodes can get route updates for forwarding data packets via different adapters for reducing interference and improving spectrum utilization.

Chapter 4

SD-MANET ROUTING

In this chapter, we present a routing protocol, called Proactive Control Communication (PCC), designed for the SD-MANET architecture described in Chapter 3. We start with the description of the protocol and then explain its communication complexity, followed by the results of the simulation experiments.

4.1 The PCC Protocol

As the name suggests, PCC is a proactive protocol, in which the SDNC proactively updates the routes in all nodes. Each node receives routes to all other nodes in the network. We describe PCC using the following three functions:

1. Learning Route to SDNC
2. Learning Network Topology
3. Sending Network Routes

The three managers inside the SDNC described in Chapter 3 perform the above three functions.

4.1.1 Learning Route To SDNC

The Connectivity Manager allows each node to learn a route to SDNC (RTS). The SDNC periodically floods a message called Topology Discovery (TD). This message includes a field (*seqNum*) for the sequence number. A new TD has a higher *seqNum* than the previous ones. The SDNC calls the *SendTD* procedure described in Algorithm 1 for sending the message. In this procedure, the sequence number is incremented by one and set to the *seqNum* field. The next call to *SendTD* is scheduled

Algorithm 1 Learning Route to SDNC

```
1: procedure SendTD
2:     TD.seqNum ← TDSeqNum++
3:     Broadcast TD
4:     Schedule (SendTD, TDInterval)
5: end procedure
6: procedure ProcessTD(TD)
7:     if TD.seqNum > savedSeqNum then
8:         savedSeqNum ← TD.TDSeqNum
9:         RTS ← TD.sender
10:        Broadcast TD
11:    end if
12:    neighbors.insert (TD.sender)
13: end procedure
```

for after *TDInterval* seconds. Nodes that receive the message call the *ProcessTD* procedure described in Algorithm 1. Nodes process the *new* TDs by checking the *seqNum* field. For new TDs, the sender becomes the RTS (i.e., the next hop on the route to SDNC). Note that the node identifies the sender from the source field in the IP header.

In addition to learning the RTS, the TD message also acts as a Hello message. So the node that receives it identifies its neighbor node (the sender) and includes it in its neighbor list. The TD flooding process is similar to the gradient routing scheme described in [121], where all nodes forward the message once and learn their reverse routes to the message originator. In this case, the SDNC is the message originator, the *seqNum* field uniquely identifies the message, and all nodes learn their route to the SDNC.

Figure 4.1 shows a network topology with SDNC flooding a TD message. At the end of the TD flooding process (1) each node learns its route to SDNC and (2) each node learns its neighbor nodes.

4.1.2 Learning Network Topology

The Topology Manager allows the SDNC to learn the network topology by collecting the neighbor information of each node. The TD flooding results in each node knowing its neighbors. Each node sends this information to SDNC in a message called

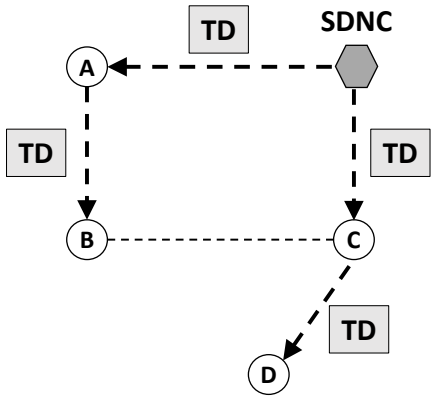


Figure 4.1: SDNC flooding a Topology Discovery (TD) message and nodes A, B, C, and D learning their route to SDNC.

Algorithm 2 Learning Network Topology

```
1: procedure SendNI
2:     NI.origin ← itself
3:     NI.neighbors ← neighbors
4:     NI.seqNum ← savedSeqNum
5:     NI.nextHop ← RTS
6:     Unicast NI to RTS
7: end procedure
8: procedure ProcessNI(NI)
9:     if NI.nextHop == itself then
10:        NI.nextHop ← RTS
11:        Unicast NI to RTS
12:    end if
13: end procedure
```

Neighbor Information (NI). Nodes call the *SendNI* procedure described in Algorithm 2 for sending the NI messages. Each node sends the message to its RTS, which then calls the *ProcessNI* procedure described in the Algorithm 2 for forwarding the received message to its RTS. On receiving the message, the Topology Manager inside the SDNC stores the neighbor information in a connectivity map, which represents the network topology as an adjacency matrix. Note that the received neighbor information is the node's link-state information.

Figure 4.2 shows node D sending its NI message to its RTS (i.e., node C) and

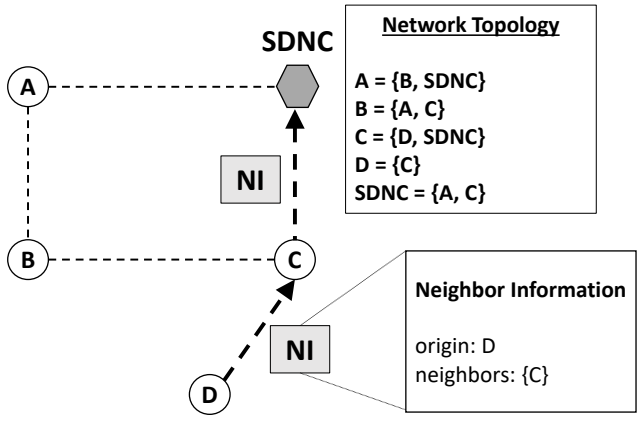


Figure 4.2: Node D sending a Neighbor Information (NI) message to SDNC via its RTS.

node C forwarding it to its RTS (i.e., the SDNC). In this topology, node D has just one neighbor (i.e., node C), so the NI message includes node D in the *origin* field and node C in the *neighbor*. Similarly, all other nodes send their NI to the SDNC via their respective RTS. In the end, the SDNC has the neighbor information of each node, representing the network topology.

4.1.3 Sending Network Routes

The routing application inside the SDNC selects routes for all nodes using the network topology collected by the Topology Manager. We have used Dijkstra's all-pairs shortest path algorithm as the routing application, but any path selection algorithm can be used. Other SDNC applications such as QoS, load balancing, and policies can also influence or determine the route selection and enable fine-grained flow-based forwarding.

The Forwarding Manager sends network routes in a message called Route Update (RU). Each node receives an RU message from the SDNC and uses the routing information in it for forwarding data packets. The RU message also includes a path for the intermediate nodes to read and forward the message. Figure 4.3 shows SDNC sending an RU message having routing information for node D and a message path.

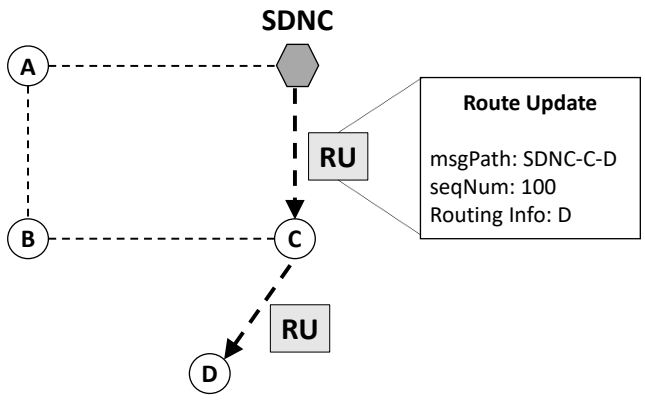


Figure 4.3: SDNC sending a Route Update (RU) message to node D.

In addition to the routing information and the message path, the RU also includes a sequence number in the *seqNumRU* field. This sequence number allows the node to acknowledge the forwarding information in the received RU message. In a mobile and dynamic environment, the RU messages sent by SDNC may not reach their respective node. One possible solution is to include the full routing information in every RU message, making nodes receive redundant information periodically. However, this approach significantly increases the communication overhead. The Forwarding Manager limits it by sending incremental forwarding information updates to the nodes. The procedure used for sending the network routes is described in Algorithm 3.

Sending incremental route updates requires nodes to send acknowledgments for the received ones. Thus, each node that receives an RU message sends a message called Route Update Acknowledgment (RUA) to SDNC by setting *ackNum* as the *seqNumRU*. The acknowledgment allows the Forwarding Manager to update the global forwarding information with the routes that were successfully delivered, as opposed to those that were sent but not delivered. In the next periodic transmission of RU, the Forwarding Manager refers to the global forwarding information and includes only the routes that have either changed or not yet acknowledged. This acknowledgment scheme reduces the communication overhead significantly. Nodes send the RUA messages in the same

Algorithm 3 Sending Network Routes

```
1: procedure SendingNetworkRoutes
2:     for each source s do
3:         for each destination d do
4:             r ← route between s and d
5:             if r ≠ previously sent OR r not acknowledged then
6:                 Include r in RU
7:             end if
8:         end for
9:         RU.seqNum ← RUSeqNum
10:        RU.msgPath ← path between SDNC and s
11:        Unicast RU
12:    end for
13:    RUSeqNum++
14: end procedure
```

manner as the NI message, i.e., via the RTS. Figure 4.4 shows node D sending an RUA message to SDNC with *seqNumRU* (from the received RU message) set to *ackNum*.

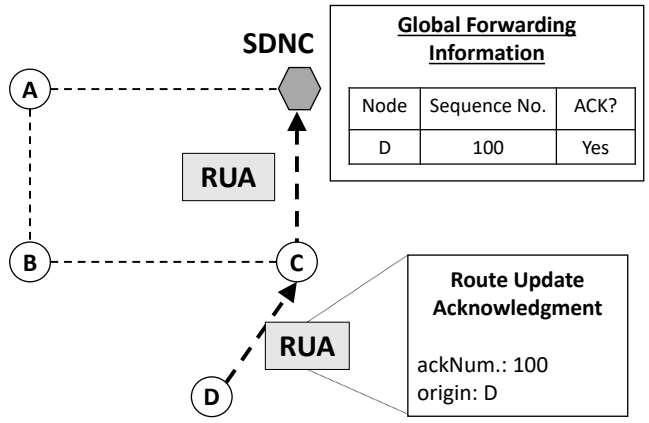


Figure 4.4: Node D sending a Route Update Acknowledgment (RUA) message to SDNC via the RTS.

4.2 Communication Complexity

We now describe PCC's communication complexity (CC). We define CC as the total bytes transmitted in the network over its entire operation. Table 4.1 lists all the

symbols used for expressing the CC. For this analysis, we assume that each transmission has a successful delivery.

Table 4.1: Symbols

Category	Symbol	Meaning
Network	**N**	Network Size
	D	Network Diameter
	M	Average Node Degree
	B	Data Packet Size
	R_{gen}	Data Packet Generation Rate
Messages	H_{td}	Topology Discovery Header Size
	H_{ni}	Neighbor Information Header Size
	H_{ru}	Route Update Header Size
	H_{rua}	Route Update Acknowledgment Header Size
Protocol	R_{td}	Topology Discovery Rate
	R_{ru}	Route Update Rate

As described in Section 4.1, PCC uses four control messages: TD, NI, RU, and RUA. Table 4.2 lists the communication complexity of each of them. Equation 4.1 represents the combined control communication complexity, where R_{td} is the topology discovery rate and R_{ru} is the route update rate.

$$CC_{ctrl\_pcc} \leq R_{td}(NH_{td} + (H_{ni} + M)ND) + R_{ru}((H_{ru} + N)ND + H_{rua}ND)) \quad (4.1)$$

We now consider the *asymptotic* control communication complexity (ACCC) of PCC. We first note that in Equation 4.1, the terms H_{td}, H_{ni}, H_{ru}, H_{rua}, R_{td}, and R_{ru} are constants. If the average node degree is d, i.e., $M = d$, the ACCC is $O(N + dND + N^2D + ND)$, which essentially is $O(dND + N^2D)$. Thus, the ACCC of PCC is:

$$CC_{ctrl\_pcc} = O(dND + N^2D) \quad (4.2)$$

Table 4.2: Communication complexity of each message

Message	Complexity	Explanation
TD	NH_{td}	SDNC broadcasts a TD message of size H_{td}, and each node rebroadcasts it.
NI	$(H_{ni} + M)ND$	Each node sends the information of its M neighbors in an NI message to the SDNC. The network diameter is D, so up to D nodes may forward.
RU	$(H_{ru} + N)ND$	SDNC sends an RU message to each node with routes to every other node in the network. Each such message gets forwarded by up to D nodes.
RU	$(H_{rua})ND$	Each node sends an RUA message with the sequence number from the received RU message. Each such messages gets forwarded by up to D nodes.

PCC being a proactive protocol, all nodes always have routes to every node in the network. When a node transmits a data packet of size B, it gets forwarded by up to D nodes (because the network diameter is D). Equation 4.3 shows the CC of sending a data packet.

$$CC_{data\_pcc} \leq BD \tag{4.3}$$

If R_{gen} is the data packet generate rate, then Equation 4.4 shows PCC's total CC.

$$CC_{pcc} \leq CC_{ctrl\_pcc} + R_{gen}CC_{data\_pcc} \tag{4.4}$$

We now discuss PCC's ACCC for two broad classes of networks: dense and sparse. For dense networks, the average node degree increases with the network size, but the network diameter remains very small, so $d = O(N)$ and $D = O(1)$. Substituting these values in Equation 4.2, we get $O(N^2 + N^2)$, which is $O(N^2)$.

For sparse networks, the node degree remains very small with an increase in network size, but the network diameter grows large, so $d = O(1)$ and $D = O(N)$. Substituting these values in Equation 4.2, we get $O(N^2 + N^3)$, which is $O(N^3)$. Table 4.3 lists PCC's ACCC for generic, dense, and sparse networks.

Table 4.3: Comparison of the asymptotic control communication complexities

	PCC
Generic	$O(dND + N^2D)$
Dense	$O(N^2)$
Sparse	$O(N^3)$

4.3 Simulation Results

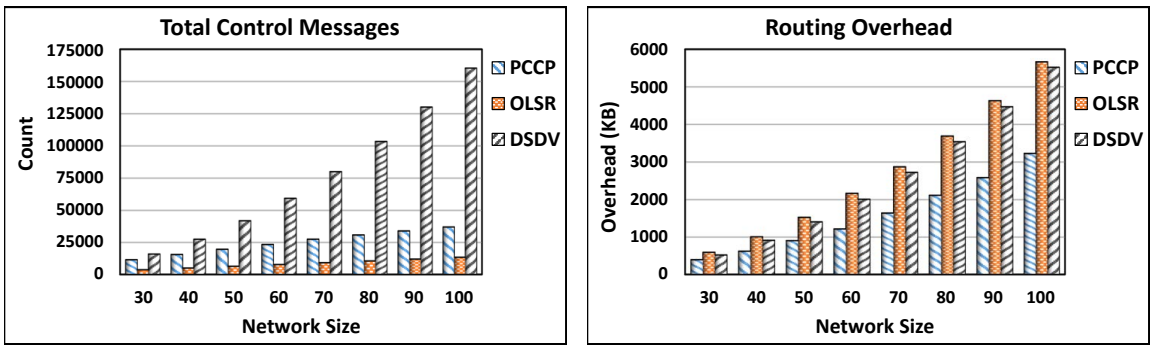
We now describe the simulation results of PCC. We have compared the results to those of OLSR and DSDV. Table 4.4 shows a complete list of simulation parameters.

Table 4.4: Simulation Parameters

Parameter	**Value**	**Parameter**	**Value**
Network size	30 to 100 nodes	Simulation runtime	200s
Simulation area	800×800m^2	Application nodes	10 pairs
Propagation loss	Friis model	Application runtime	50s to 200s
Mobility	Random waypoint	Applications rate	48Kb/s
Node speed	5m/s	Packet size	1024B
RD, NI, and RU Intervals	2s, 2s, and 3s, resp.	MAC	802.11B
DSDV full update	15s	OLSR Hello and TC	2s and 3s

The metrics used for comparing the results are (1) Total Control Messages (TCM), which is the total number of control messages forwarded by the nodes, (2) Routing Overhead (RO), which is the cumulative size of all the control messages forwarded over the lifetime of the simulation, (3) Packet Delivery Ratio (PDR), which

is the ratio of data packets successfully delivered to data packets sent, and (4) Average Delay (AD), which is the average time taken by a data packet to reach to its destination. The results are an average of 20 randomly seeded runs.



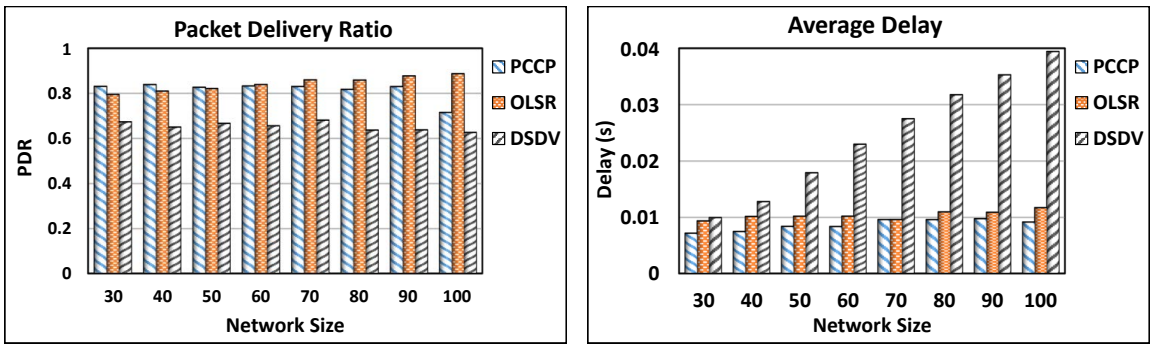
(a) Total Control Messages

(b) Routing Overhead

Figure 4.5: Results showing Total Control Messages and Routing Overhead.

Figure 4.5(a) shows PCC sending more control messages than OLSR but fewer than DSDV. For network size 100, PCC sends ∼2.7x more control messages than OLSR and ∼4.3x fewer than DSDV. Here, DSDV sends the most number of control messages because of its triggered updates for every change in the routing table in addition to the periodic updates. PCC forwards more control packets than OLSR because it collects neighbor information from all nodes and sends route updates to all nodes. As a result, several nodes have to forward the control packets to and from SDNC. However, Figure 4.5(b) shows PCC having up to 1.7x lower (better) routing overhead than both DSDV and OLSR because PCC sends incremental route updates when the SDNC identifies a change in network topology and different routes are available or when the previously sent ones are not acknowledged. By contrast, OLSR and DSDV periodically exchange large link-state and distance-vector information, respectively, among all neighbor nodes and incur more routing overhead.

Figure 4.6(a) shows PCC having a better PDR than both OLSR and DSDV for networks of size up to 50 nodes, but as the network size increases, the PDR drops. This



(a) Packet Delivery Ratio

(b) Average Delay

Figure 4.6: Results showing Packet Delivery Ratio and Average Delay.

drop is a result of increasing congestion because the SDNC receives and transmits more control packets. The increasing congestion results in several dropped NI messages and the SDNC failing to learn the complete network topology. As a result, the SDNC fails to select and send route updates to nodes. Without the route updates, nodes drop several data packets, resulting in a low PDR. Figure 4.6(b) shows PCC having almost the same AD as OLSR but up to 4x lower than DSDV. Nodes transmit numerous control packets in DSDV (shown in Figure 4.6(a)), resulting in far more MAC layer back-offs and retransmission and higher average delay for the data packets.

4.4 Conclusions

In this chapter, we described a centralized proactive protocol called PCC for our SD-MANET architecture. The architecture is described using the three network functions described in Chapter 3. We then explain PCC's communication complexity and simulation results.

The simulation results indicate that a centralized routing protocol is appropriate only for small networks (up to network size 50). We have identified two reasons that limit PCC's performance. They are:

1. Communication Overhead

- Learning network topology using the neighbor information of all nodes results in several NI message forwardings because each NI message gets forwarded by all intermediate nodes between the source and the SDNC.
- Individually sending route updates to each node results in several RU forwardings because each RU message gets forwarded by all intermediate nodes between the SDNC and the destination.

2. Unreliable Transmission

   - Congestion at the SDNC and unreliable transmission of NI messages may result in SDNC not learning the full network topology, and in some cases learning a *disconnected* topology, resulting in SDNC failing to select routes for some of the nodes in the network.
   - Failure to select routes for some of the nodes results in those nodes not receiving routing information and failing to forward data packets to their destination.

Chapter 5

ECHO

In this chapter, we present a protocol, called ECHO, designed for network-wide broadcasting in MANETs. This protocol caters to the requirements of low-rate long-range networking discussed in Chapter 2.

Most, if not all, routing protocols use control packets for collecting local or global topology information. However, there are several contexts in which the network capacity is so low that control packets overwhelm the system by themselves. For example, in many disaster relief, public safety, and IoT networks, it is of paramount importance to have *connectivity*. The number of nodes required to provide connectivity in an area increases super-linearly with decreasing range. Our simulations have shown that with a 1-mile (1600m) transmission range only 25 nodes are required to connect a 3 mile^2 (4.8 km^2) area, but upwards of 1000 nodes are needed if the range is 200m (WiFi range)[95]. The high deployment cost of short-range devices has motivated the development of long-range technologies, such as LoRa [10] and SigFox [15], and products like goTenna [4]. However, long range implies low data rate (all other factors held constant) – e.g., LoRa chips have a data rate of 27 kbps [10]. Control packet overhead in such systems is prohibitively expensive. Further, control packets may cause network instability under lossy conditions.

The ECHO protocol constructs and maintains a broadcast backbone without using any control packets. Instead, using a field in the data packet header, a node listens for an "echo" of the specific packet that the node transmitted to determine its membership in the backbone. ECHO is deterministic, source-independent, fully distributed, accommodates mobility, and balances battery consumption across nodes.

We prove formally that the broadcast backbone that ECHO produces is sufficient for a source-independent network-wide broadcast.

We first explain the problem of network-wide broadcast along with its existing solutions and then describe the ECHO protocol. Later, we prove the correctness of ECHO and analyze its communication complexity, comparing it with that of Flooding [87] and MPR [94]. In the end, we present the results of simulation experiments for networks of varying size, density, network load, node speed, and data rate.

5.1 Network-Wide Broadcast

In a MANET, it is often necessary to do a Network-Wide Broadcast (NWB), that is, send a packet from a given source to all nodes in the network. Examples include position updates for collaborative mapping, situation reports, group chats, clock synchronization messages, and routing control messages [35, 92].

A simple solution to the NWB problem is *Flooding*, namely having every node retransmit the message once. However, this results in excessive transmissions and collisions, causing what is commonly known as a *broadcast storm* [110]. This has motivated several efforts toward *efficient* NWB, that is, reducing the total number of transmissions [79, 89, 105, 33, 94, 80, 32, 99]. Most if not all of these works are either probabilistic (i.e., do not guarantee delivery even in lossless conditions), assume location information, or utilize control packets to collect local or global topology information [108]. A comparison and classification of solutions into probability-based, area-based, and neighbor-knowledge methods is given in [116].

The NWB problem is to determine the (minimal) set of nodes that should re-transmit the packet so that it reaches all nodes in the network. From a graph-theoretic viewpoint, the network-wide broadcasting problem in a centralized setting can be formulated as the *Minimum Connected Dominating Set* problem, which is NP-complete [56]. An $O(H(d))$ approximation algorithm (d is the maximum degree, and H is the harmonic number) is given in [58].

Location-based, and counter-based schemes are described in [110], and probabilistic schemes are studied in [105, 33, 82]. These schemes reduce the number of transmissions significantly but are not reliable, i.e., do not guarantee delivery even in lossless conditions [108]. Deterministic schemes, such as multi-point relaying (MPR) [94] and dominant pruning [80, 32], are based on covering a two-hop neighborhood with a minimal set of one-hop neighbors. The IETF standard OLSR [35] uses multi-point relaying schemes [94, 93]. These schemes are *source dependent*, i.e., each node selects its relay nodes using the collected topology information, and hence, the relay nodes vary based on the sender of the packet. Further, nodes either include the relay node ids in the message itself or inform them via control packets. *Source independent* methods were proposed in [107, 118], but they all make extensive use of control packets.

Existing protocols are either probabilistic or they obtain topology information via control packets, making their application limited and unreliable in low-capacity networks. We present the first deterministic, zero-control-packet, location-unaware protocol for efficient network-wide broadcasting in mobile multi-hop wireless networks. Called ECHO, our protocol uses node identifier information within the data packet header to determine – in a fully distributed and source-independent manner – the set of *critical*[1] nodes whose transmission is sufficient for a network-wide broadcast.

ECHO consists of two interwoven phases: Full Flood (FF) and Pruned Flood (PF). The FF phase executes periodically, flooding a randomly chosen data packet and selecting critical nodes. Specifically, a node marks itself critical if and only if it receives an "echo" of its transmission, i.e., the node receives a packet with itself identified as the previous sender. The PF follows the FF phase, wherein only the critical nodes rebroadcast. An FF data packet originated at a single node builds critical nodes that are valid for broadcasting packets originated from any node. That is, the selected critical nodes are *source independent*. Nodes transmit an overwhelming majority of packets via PF until the next FF, resulting in highly efficient network-wide broadcasts.

[1] Also referred to in the literature as *dominating*, *relay* or *rebroadcast* nodes.

Unlike prior deterministic protocols, ECHO does not use any control packets or explicit topology information.

5.2 The ECHO Protocol

ECHO is a network-layer protocol that efficiently delivers an application-layer message to all reachable nodes in the network. Data packets are prefixed with a header consisting of the following relevant fields. The descriptions are with respect to a considered node, say C.

- *origin*: The node that originated the packet, its "source".
- *sender*: The node from which C received the packet.
- *prevSender*: The node from which the *sender* first received the packet (N/A if sender is origin)
- *seqNum*: A sequence number unique at the origin.
- *floodIndicator*: A 1-bit field to indicate if this packet is a Full Flood (FF) or Pruned Flood (PF).

A data packet received from the transport layer is marked either *Full Flood (FF)* or *Pruned Flood (PF)*. The procedure for this marking is described in Section 5.2.2 via the *flood-indicator*. A packet marked FF is sent using Flooding, i.e., transmitted exactly once by all nodes. During an FF, each node running ECHO determines if it should mark itself *critical* or not; and during PF, only critical nodes forward. Thus, the core part of ECHO happens in the FF phase where critical nodes constituting a broadcast backbone are selected distributively. FF packets are sent occasionally to reset the critical nodes to account for topology changes. A vast majority of packets are forwarded in the PF mode where only critical nodes forward. Figure 5.1 illustrates a rough overview of the ECHO operation.

Thus, ECHO has two key parts: (1) determining critical nodes based on a single flood, and (2) managing the periodic floods to refresh the critical nodes to account for topology changes. We describe each of them in the following subsections.

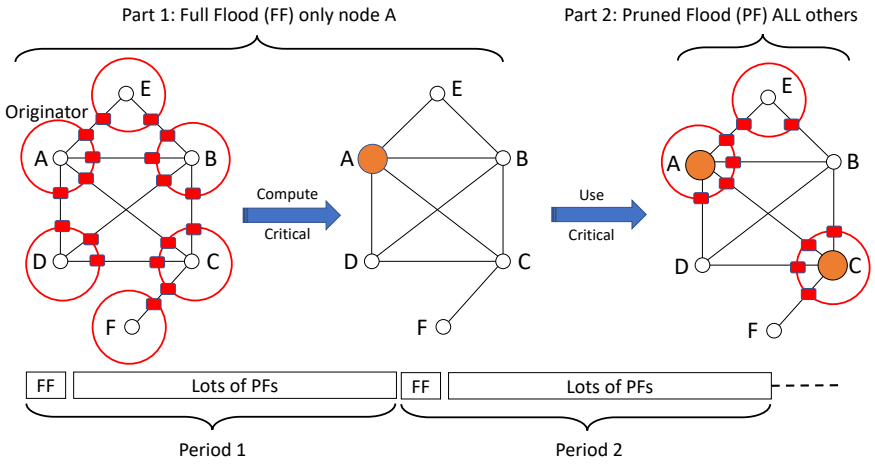


Figure 5.1: ECHO picks a random packet periodically to flood (Full Flood). This data packet is used to select critical nodes. After that, only critical nodes relay the data packets (Pruned Flood). In this example, node A originates a data packet. This is marked FF and flooded. All nodes use Algorithm 4 (see Section 5.2.1) to compute critical node. Thereafter, all packets are relayed only by critical nodes

5.2.1 Determining Critical Nodes

Upon receiving a data packet, marked FF, a node C first determines if it is a duplicate by referring to the *seqNum* field, as in Flooding. If it is not, then C retransmits the packet, but before doing so, sets the *sender* field to C and the *prevSender* field to the *sender*. If it is a duplicate, then unlike Flooding, which discards the packet, C checks if the *prevSender* field is its id, namely C. If it is, we say that "C hears an echo", but it would be true if the neighbor first received the packet from C. If C hears an echo of its transmission, then C marks itself "critical". However, if this condition is not met by *any* packet for a configured period (timeout), then C marks itself non-critical. Figure 5.2 shows a state diagram for this aspect of ECHO.

Intuitively, if the packet was not echoed, then it means that the packet was a duplicate for all neighbors – and that in turn means all of its neighbors can receive the packet from some other node, and hence, this node need not forward subsequent packets, i.e., be critical. We note that the originator of FF is always a critical node.

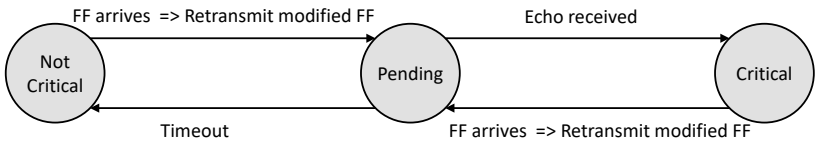


Figure 5.2: ECHO critical node computation state diagram. "Echo Received" means receiving a Full Flood with previous-sender marked as own id.

The ECHO distributed algorithm for determining critical nodes is given in Algorithm 4. We show the key pieces of the logic, i.e., steps used by nodes for marking themselves either critical or non-critical. The algorithm does not include the steps for randomizing the FF generation. Upon completion of an FF phase, if a node receives a data packet marked PF, it checks if its state is critical or pending. If so, then it re-transmits the packet, else it does not. Figure 5.3 illustrates an example execution of ECHO.

Algorithm 4 ECHO Algorithm at node C

```
1:  procedure WHENPACKETRECEIVED(pkt)
2:      if pkt.seqNum has not been received earlier then
3:          pkt.previous-sender ← pkt.sender
4:          pkt.sender ← C
5:          Rransmit pkt
6:          Set echo timer                    ▷ Round-trip delay + margin
7:      else if pkt.previous-sender equals C then
8:          Mark CRITICAL                     ▷ Echo received
9:          Discard pkt
10:         Clear echo timer
11:     end if
12: end procedure
13: procedure WHENECHOTIMEREXPIRES
14:     Mark NON-CRITICAL                     ▷ No echo received
15: end procedure
```

5.2.2 Managing Full-Flood Generation

Data packets across all nodes are marked PF by default and FF approximately once every *FFInterval*, which is configured based on expected topology dynamics. It is

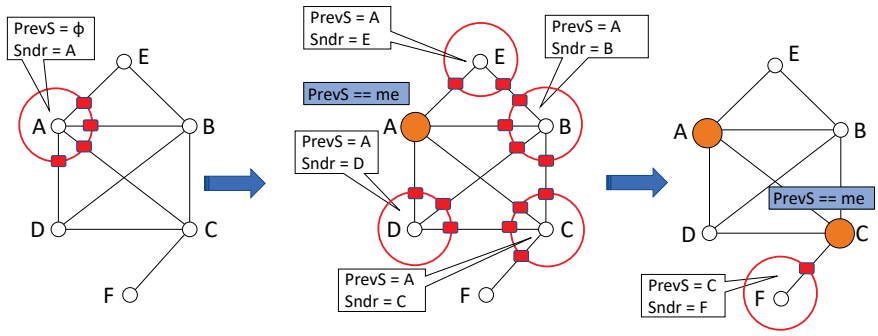


Figure 5.3: Example ECHO operation on FF originating from node A. Since nodes A and C are the only nodes that receive an "echo" (previous-sender equals identifier), they mark themselves critical (big filled circle) and the others mark themselves non-critical. Subsequent packets are forwarded only by nodes A and C irrespective of originator (source independence).

done in a fully distributed manner as follows. Each node, upon receiving a data packet from the application, checks if it has received an FF within the last *FFInterval*. If not, the node marks the packet as FF, and all nodes execute the critical node determination as described in Section 5.2.1. Note that this suppresses the generation of FF from other nodes in most cases. To further suppress overlapping and simultaneous FFs, a node that originates the FF uses a longer interval for its next FF, that is, α***FFInterval*. In our implementation, we have used the value of α as 3. We note that in an unlikely scenario of multiple nodes originating FFs simultaneously, ECHO continues to maintain its correctness. The overlapping FFs would result in the selection of a few additional critical nodes but ensure network-wide broadcast during PF. Further, the FF originator is random, so the set of critical nodes selected may be different from the previous one yet sufficient for network-wide broadcasts. This randomization helps balance energy consumption across nodes

Between two FFs, topology changes may cause loss in delivery. However, our experiments over random mobile topologies have shown that the loss is very much tolerable (see Section 5.5).

5.2.3 Overhead

A natural question is whether ECHO's Full Floods negate any gains from eliminating control packets. We note that the Full Flood only happens for one data packet within the FF interval. Thus, if we have 30 nodes, and the FF and origination intervals are 60 and 30 seconds, respectively (the simulation parameters in Section 5.5), there is only one FF for every 60 PFs – a negligible fraction. This ratio increases with decreasing origination interval and increasing size.

One may also argue that while there are no control packets, some additional control-like information, namely the previous sender, which is typically not part of the header, is used. We note that while the *prevSender* information is an addition to the header and included in every data packet, its size is not equivalent to the size of an entire control packet and remains constant independent of network size or density. Although it is technically 2-hop-away information, it is not tantamount to a 2-hop *topology* because it accounts for only a single link between the sender and one of its neighbors, i.e., the previous sender. Specifically, if the average node degree is d, there is only $O(2d)$ information and not $O(d^2)$, and even this information is not explicitly stored or used.

Further, including this information as part of the header is much more efficient than placing it in a separate control packet as the latter will incur all of the MAC and PHY-layer header overhead for each packet. In the results section, we show that the overhead due to the *prevSender* information is insignificant compared to that of the control packets in other approaches.

5.3 Theoretical Analysis

We first present a proof of correctness of ECHO. Then, in Section 5.4, we derive ECHO's communication complexity and compare it with that of MPR.

Given a graph G=(V, E), we show that Algorithm 4 running on each node will, in the FF phase, result in a *Connected Dominating Set (CDS)*: a subset of *dominating* nodes such that the CDS is a connected subgraph of G, and every node in G is either

in the CDS or adjacent to at least one node in the CDS [58]. It is easy to see that a CDS is necessary and sufficient for guaranteed broadcast delivery barring packet losses. Below, we use the terms "critical" and "dominating" interchangeably.

For this section, we make three assumptions: the graph G is connected (A1); messages are ordered on receive, i.e., there is a notion of "first received" message (A2); and the system is lossless (A3). We note that these are for the proof only, our simulations (Section 5.5) include packet losses and network partitioning. Assumption A2 is true for most if not all real systems, and assumptions A1 and A3 are only for simplifying the proof – our simulations include disconnected networks and packet losses. The discussion below is for a single Full Flood packet – we show that at the end of the flood a CDS is formed.

Definition 5.3.1. *Given a node u, let $p(u)$ denote the node from which u first received the packet. Let $c(u) = \{v\colon p(v) = u\}$.*

The $p(u)$ and $c(u)$ denote, respectively, the "parent" and set of "children" of u. Note that $c(u)$ can be an empty set, and $p(u)$ is undefined for the originator.

Observation 5.3.1. *The set of edges $(u, p(u))$ constitute a spanning tree of G rooted at the originator of the flood.*

To see this, note that by Algorithm 4 every node transmits the packet once. From assumptions A1 and A3, every node receives the message, and by A2 there is a unique $p(u)$ for every u.

Lemma 5.3.2. *ECHO marks a node x critical if and only if $|c(x)| > 0$.*

Proof. Consider a node x. If $|c(x)| > 0$, there exists some child y of x. Per Definition 5.3.1, y received the packet first from x. In the execution of ECHO, y will invoke lines 3-6 in Algorithm 4. The transmission will be received by x per assumption A3. In the execution of ECHO at x, lines 8-10 will be executed as x has already received this packet (line 2). Hence, node x is marked critical. For the "only if" case, if x is marked

critical, then it must have received a packet from some y with previous-sender equal to itself, implying that y received the packet first from x. This in turn implies, per definition 5.3.1 that $x = p(y)$, hence $|c(x)| > 0$. □

By Lemma 5.3.2, all nodes except those without children, namely the "non-leaf" nodes, are critical nodes. Since the originator is reachable from every critical node via a chain of parent nodes, we have

Observation 5.3.3. *The set of critical nodes induce a connected subgraph.*

Lemma 5.3.4. *Every node with $|c(x)| = 0$ ("leaf" nodes) is adjacent to a critical node.*

Proof. Let x be a leaf node. Let $y = p(x)$. Clearly, y has a child, i.e., $|c(y)| > 0$. By Lemma 5.3.2, y is a critical node, and since a parent is adjacent to the child, the lemma follows. □

The following theorem combines the above lemmas and proves the correctness of ECHO.

Theorem 5.3.5. *For any connected network $G = (V, E)$, and a given packet flooded from a node, the critical nodes as per Algorithm 4 form a connected, dominating set.*

Proof. Every node is eventually either critical or non-critical. Per Lemma 5.3.4, if a node is not critical, it is adjacent to a critical node. Therefore the set of critical nodes is a *dominating* set. Per Observation 5.3.3, the critical nodes are *connected*. Thus, Algorithm 4 results in a connected dominating set. □

Neither Algorithm 4 nor the proof of Theorem 5.3.5 utilized the origin (source) of the packet, and a PF originated at any node can be delivered with only critical nodes transmitting. *Thus, ECHO is source independent.*

We note that while ECHO ensures that the critical nodes are chosen so that network-wide broadcast reaches all nodes, it does not guarantee optimality. This is not surprising since the Minimum Connected Dominating Set problem is NP-complete even in the centralized setting [58], and therefore polynomial-time optimal algorithms are highly unlikely.

5.4 Communication Complexity

We now consider the communication complexity CC, that is the number of bytes per second for ECHO, Flooding, and Multi-point Relays [94], a well-known method for reducing broadcast transmissions, and used in the OLSR [35] protocol. While several implementations of OLSR exist where MPR is used, we are not aware of any implementation or model of MPR by itself. The reason we need a stand-alone implementation of MPR is that OLSR disseminates topology updates which are not needed for MPR. Therefore, we have implemented MPR ourselves as described briefly below.

Table 5.1: Symbols

Category	Symbol	Meaning
Network	**N**	Network Size
	D	Network Diameter
	M	Average Node Degree
	B	Data Packet Size
	R_{gen}	Data Packet Generation Rate
	N_r	MPR Relay Nodes
	N_c	ECHO Critical Nodes
Messages	b_1	Flooding Header Size
	b_2	ECHO Header Size
	b_3	MPR Header Size
Protocol	R_{ff}	ECHO Full Flood Rate
	R_h	MPR Hello Rate

Each node transmits a Hello packet every h seconds by including its neighbor list. The link status from each neighbor is also included, and it could be either asymmetric, symmetric, or MPR. The nodes that receive the packet perform the following operations: (1) add the sender to their neighbor list, (2) update their two-hop neighborhood information with the neighbor list of the sender, (3) select their MPR relay nodes using the two-hop neighborhood information and the heuristic algorithm described in RFC 3626 [35], and (4) learn the MPR relay nodes selected by the sender.

The MPR relay nodes thus determined are responsible for forwarding the packets sent by the selector node.

Table 5.1 shows all symbols used in the CC analysis. Let N be the number size and B be the data packet size in bytes. Let b_1, b_2, and b_3 denote, respectively, the Flooding header, ECHO header, and MPR header size in bytes.

We assume that each node periodically generates broadcast traffic at a rate of R_{gen} packets per second. We assume that all protocols use Broadcast transmission at the MAC layer, which means that a single transmission is sufficient for all neighbors. All expressions below relate to the complexity of a generation *cycle*, that is, R_{gen} packets generated from *each* node.

For Flooding (FLDG), each node generates $R_{gen}(B+b_1)$ bytes, and each packet is transmitted once by every other node. Thus,

$$CC_{FLDG} = R_{gen}N^2(B+b_1) \tag{5.1}$$

For ECHO, suppose the Full Flood frequency is $R_{ff} <= R_{gen}$, and let N_c denote the number of critical nodes. Then,

$$CC_{ECHO} = R_{ff}N(B+b_2) + R_{gen}N(N_c+1)(B+b_2) \tag{5.2}$$

Here, the first term is the complexity of the Full Flood. The second term captures the complexity of Pruned Floods being sent only by the critical nodes and the originator.

Finally, for Multi-Point Relays, if R_h is the rate of sending Hello packets, M is the average number of neighbors, and N_r is the number of relaying nodes, then,

$$CC_{MPR} = R_hMN + R_{gen}NN_r(B+b_3) \tag{5.3}$$

Here, the first term is the control overhead, and the second term is the cost of transmitting data packets. We note that this matches the expression derived in [68]

for the MPR part, that is, excluding the global link-state updates which are present in OLSR but not in MPR.

We now consider the relative communication complexity of ECHO with respect to Flooding and MPR with a view to gaining insight on what influences it. From Equations 5.1 and 5.2, the ratio of the complexities of Flooding and ECHO is:

$$\frac{CC_{FLDG}}{CC_{ECHO}} = \frac{(B+b_1)}{(B+b_2)}\left[\frac{N}{\frac{R_{ff}}{R_{gen}}+N_c+1}\right] \tag{5.4}$$

Thus, the relative gain over Flooding increases with decreasing R_{ff} and decreasing N_c. The slower the topology changes, the smaller we can make R_{ff}, and increase the advantage of ECHO over Flooding.

Similarly, from Equations 5.2 and 5.3, the ratio of expected transmissions for MPR and ECHO is:

$$\frac{CC_{MPR}}{CC_{ECHO}} = \frac{(B+b_3)}{(B+b_2)}\left[\frac{\frac{R_h M}{(B+b_3)}+R_{gen}N_r}{R_{ff}+R_{gen}(N_c+1)}\right] \tag{5.5}$$

Thus, the relative gain over MPR increases with increasing R_h and increasing average number of neighbors M.

We now consider the *asymptotic* communication complexity for each of the protocols. We first note that in Equations 5.1, 5.2, and 5.3, the terms R_{gen}, R_{ff}, B and b_i are constants. Therefore, the asymptotic communication complexity of Flooding is $O(N^2)$ and of ECHO is $O(N + NN_c)$, which is $O(NN_c)$. For MPR, the average number of neighbors M is same as d (average node degree), and hence the complexity is $O(Nd + NN_r)$.

We consider two broad classes of networks: *dense* and *sparse*. By dense networks, we mean networks where the node degree $d = O(N)$, i.e., d increases as the network grows. Growth regimes where the area of deployment is held constant fall into this class, and includes fully connected (or nearly so) "parking lot" networks. In such networks, the critical/relay nodes (N_c for ECHO and N_r for MPR) are nearly constant ($O(1)$) no matter the size of the network as each such node can reach $O(N)$ nodes.

Substituting these values in the asymptotic expressions in the previous paragraph, the asymptotic communication complexity of MPR is $O(N^2+N)$ which is $O(N^2)$, and that of ECHO is $O(N+N)$, which is $O(N)$.

By sparse networks we mean those where $d = O(1)$, i.e., d is constant as N increases. Growth regimes where the area of deployment increases with N fall into this class, and includes "tree-like" networks. In such networks, the number of relay nodes (N_c and N_r) have to grow as $O(N)$ no matter the protocol. Thus, the complexity of both ECHO and MPR is $O(N^2)$.

To summarize, the asymptotic communication complexity of Flooding, MPR and ECHO for generic, dense, and sparse networks are as given in rows 2, 3, and 4, respectively of Table 5.2. We note that these values are for R_{gen} packets generated per node, and not for a single packet.

Table 5.2: Communication complexity

	Degree	Flooding	MPR	ECHO
GENERIC	d	$O(N^2)$	$O(Nd+NN_r)$	$O(NN_c)$
DENSE	$d = O(N)$	$O(N^2)$	$O(N^2)$	$O(N)$
SPARSE	$d = O(1)$	$O(N^2)$	$O(N^2)$	$O(N^2)$

ECHO's communication complexity is $O(N)$ lower (better) than that of MPR (and other topology based protocols) and Flooding in dense networks. Intuitively, the key reason for the difference is the control traffic, in particular the cost of conveying the 2-hop topology information which does not scale. Since dense networks are quite common in military deployments, the reduction of complexity in dense deployments is a crucial and unique advantage of ECHO.

5.5 Simulation Results

We have implemented and evaluated ECHO, Flooding, and MPR in ns3. Table 5.3 lists the scenarios considered for the evaluation. In the first four scenarios,

nodes transmit at a 25 kbps data rate to model low-capacity networks, whereas in the fifth scenario, nodes transmit at a 1 Mbps data rate. Table 5.4 lists the simulation parameters considered for the evaluation.

Table 5.3: Simulation Scenarios

Simulation Scenario	**Size** (nodes)	**Density** (nodes/ km^2)	**Packet Interval** (secs.)	**Node Speed** (m/s)	**Data Rate**
Increasing Network Size	[10, 100]	3.3	30	4	25 Kbps
Increasing Network Density	100	[2.77, 25]	30	4	25 Kbps
Increasing Network Load	30	3.3	[5, 0.5]	4	25 Kbps
Increasing Node Speed	100	3.3	30	[4, 20]	25 Kbps
Increasing Network Size	[10, 100]	3.3	30	4	1 Mbps

Table 5.4: Simulation Parameters

Parameter	**Value**	**Parameter**	**Value**
Simulation Time	60 mins	Node Speed	4 m/s
Data Packet Size	50 Bytes	Node Mobility	Rand. Waypoint
Propagation Loss	Friis Model	Trans. Power	15 dBm
ECHO			
FF Interval	60 secs	Timeout	0.5 secs
MPR			
Hello (MPR-2)	2 secs	Hello (MPR-60)	60 secs

We have used two different Hello intervals for MPR, namely, 2 and 60 seconds, and the graph plots show their results as MPR-2 and MPR-60, respectively. The MPR-based routing protocols, such as OLSR, use 2 seconds as the default interval. However, we have found that the performance of MPR improves in our settings if we use large intervals, such as 60 seconds.

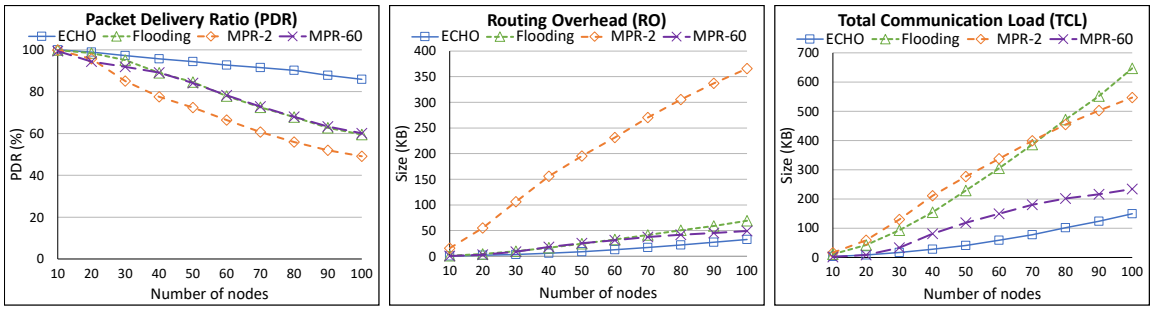
As shown in Table 5.4, the Full Flood (FF) Interval used in the ECHO simulations is 60 seconds, so the Hello interval of 60 seconds in the MPR simulations makes

R_h equal to R_{ff}. A larger Hello interval results in fewer Hello (control) packet transmissions, causing less overhead and interference but also results in less frequent link discoveries and selection of relay nodes.

The simulation results are compared using the following three metrics: (1) Packet Delivery Ratio (PDR), which is the ratio of the total data packets received and transmitted, (2) Routing Overhead (RO), which is the cumulative size of the data packet headers and the control packets (in MPR), sent per minute, and (3) Total Communication Load (TCL), which is the total bytes sent per minute, comprising of the data packet headers, control packets (in MPR), and the payload sizes. All these metrics are standard measures for efficient broadcast evaluation [116, 108]. The comparison shown is an average of 10 randomly seeded runs.

Note that we describe all numerical comparisons with the better-performing MPR-60 unless mentioned otherwise.

5.5.1 Increasing Network Size



(a) Packet Delivery Ratio (b) Routing Overhead (c) Total Communication Load

Figure 5.4: Simulation results for the increasing network size scenario where the size ranges from 10 to 100 nodes but the density remains constant.

Figure 5.4 presents the simulation results for networks having sizes ranging from 10 to 100 nodes. As the network size increases, Flooding and MPR endure a steep drop in the PDR, whereas ECHO's drop is moderate (shown in Figure 5.4(a)). For the network size 100, ECHO's PDR is ∼1.4x better than both Flooding's and MPR's.

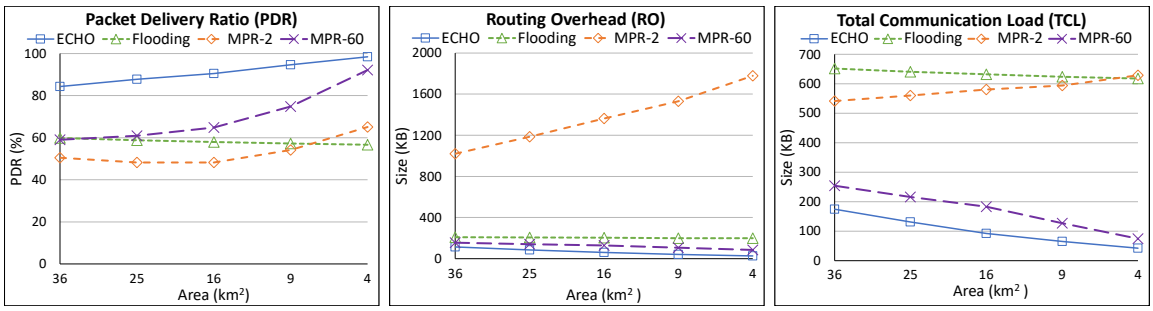
Flooding requires each data packet to be forwarded by every node in the network, resulting in several redundant transmissions. MPR requires each node to transmit a Hello packet every h seconds containing the neighbor list, resulting in transmissions of several large-sized control packets. ECHO uses no control packets and only the source-independent critical nodes forward during the PF phase, reducing the packet forwardings significantly. Moreover, nodes transmitting at a 25 Kbps data rate experience long transmission delays, making the network susceptible to packet losses due to increased congestion, interference, and packet collisions. ECHO's relative gain over Flooding and MPR in the communication complexity (described in Section 5.4) helps to keep the packet collisions and interference low and to attain a high PDR.

Figure 5.4(b) shows ECHO having the lowest RO, which is ∼2.1x lower than Flooding and ∼1.5x lower than MPR for network size 100. The RO in ECHO accounts for the size of the header fields described in Section 5.2. Although Flooding has fewer header fields (i.e., only *seqNum* and *origin*) than ECHO, it results in a higher RO because of the redundant transmissions. Despite R_h being equal to R_{ff}, MPR's RO is more than ECHO's because the Hello packets in MPR cause more overhead than the FF procedure in ECHO.

Flooding's and MPR's TCLs are also significantly higher than ECHO's because the TCL accounts for the RO and the payload of each data packet forwarded. Figure 5.4(c) shows ECHO having ∼4.3x lower TCL than Flooding and ∼1.5x lower TCL than MPR for network size 100.

5.5.2 Increasing Network Density

Figure 5.5 presents the simulation results for networks having size 100 and densities ranging from 2.7 to 25 nodes/km^2. Figure 5.5(a) shows that as the density increases, the PDR drops for Flooding. The drop is due to the growing packet collisions and interference. On the other hand, PDR improves for both ECHO and MPR because they both select fewer nodes for relaying the data packets. However, ECHO's PDR is ∼1.4x better than both MPR's and Flooding's for the lowest density (i.e., area 36



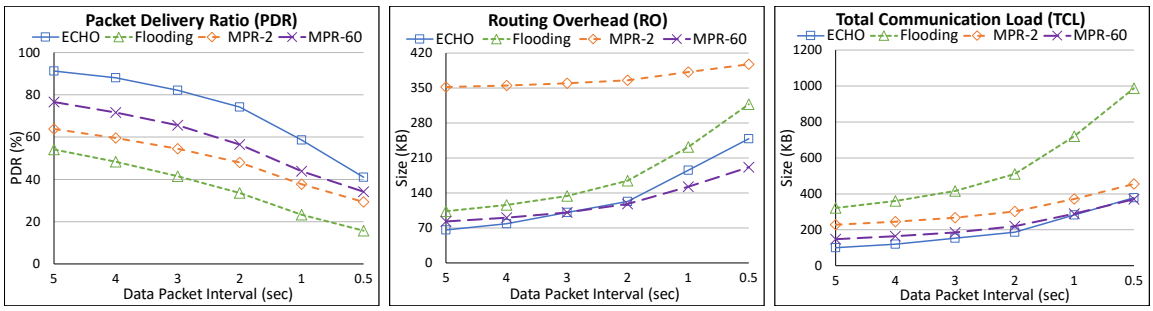
(a) Packet Delivery Ratio (b) Routing Overhead (c) Total Communication Load

Figure 5.5: Simulation results for the increasing density scenario where the network size is 100 nodes but density ranges from 2.77 to 25 nodes/km^2 (i.e., simulation area ranges from 36 to 4 km^2).

km^2). The PDR of MPR is lower because the large-sized Hello packets used for selecting the source-dependent relay nodes are affected the most by the increasing interference and collisions. All protocols suffer from low PDRs in sparse networks because of two reasons: (1) intermittent links and disconnected topology, and (2) the classic Hidden Terminal problem. However, ECHO attains ∼1.4x better PDR than both Flooding and MPR for the lowest density (i.e., area 36 km^2).

Figure 5.5(b) shows Flooding having almost the same RO for all densities because the network size remains the same (i.e., 100 nodes), and hence, the same number of data packets are forwarded. RO reduces marginally for both ECHO and MPR in dense networks because fewer nodes forward the data packets. However, ECHO has ∼3x lower RO than MPR and ∼7x lower RO than Flooding for simulation area of 4 km^2.

Figure 5.5(c) shows that similar to the RO, the TCL reduces for both ECHO and MPR because fewer nodes forward the data packets. TCL also reduces marginally for Flooding because of its decreasing PDR. However, ECHO has ∼1.7x lower TCL than MPR and ∼14x lower TCL than Flooding for area 4 km^2, whereas it has ∼1.4x lower TCL than MPR and ∼3.7x lower TCL than Flooding for area 36 km^2.



(a) Packet Delivery Ratio (b) Routing Overhead (c) Total Communication Load

Figure 5.6: Simulation results for the increasing network load scenario where the network size is 30 nodes but the data packet interval ranges from 5 to 0.5 seconds.

5.5.3 Increasing Network Load

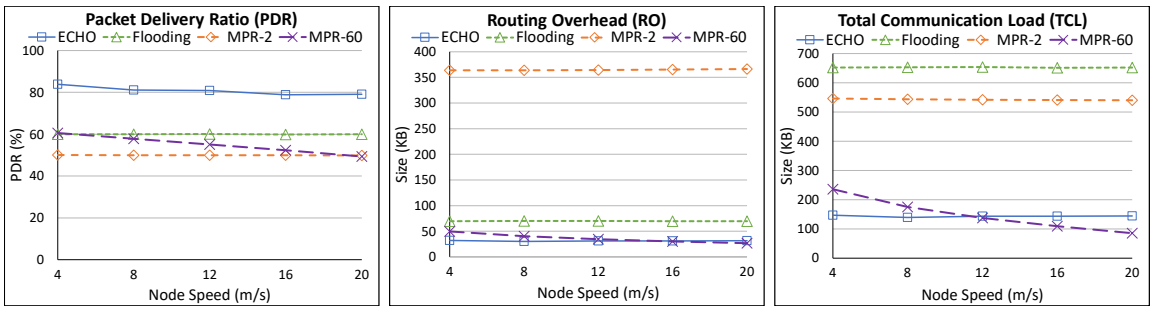
Figure 5.6 presents the simulation results for different network loads (i.e., data packet intervals). Figure 5.6(a) shows the PDR dropping steeply for all protocols because the network load gets too extreme for a 25 Kbps data rate. However, ECHO continues to provide the best PDR, which is ∼1.2x better than MPR and ∼1.7x better than Flooding.

Figure 5.6(b) shows RO increasing for all protocols because nodes generate more data packets at smaller intervals. Flooding has the highest RO because of its $O(N^2)$ complexity. At higher network loads, ECHO causes more RO than MPR because it provides a better PDR than MPR and forwards more data packets, both contributing to large RO.

The TCL includes the payload, so Figure 5.6(c) shows a steep increase in the TCL for Flooding. TCL also increases for both ECHO and MPR, and it is almost the same for both at high loads.

5.5.4 Increasing Node Speed

Figure 5.7 presents the simulation results for networks having size 100 nodes and node speeds ranging from 4 to 20 m/s. Figure 5.7(a) shows that as the node speed increases, ECHO endures a moderate drop is the PDR, whereas MPR's drop is steep.



(a) Packet Delivery Ratio (b) Routing Overhead (c) Total Communication Load

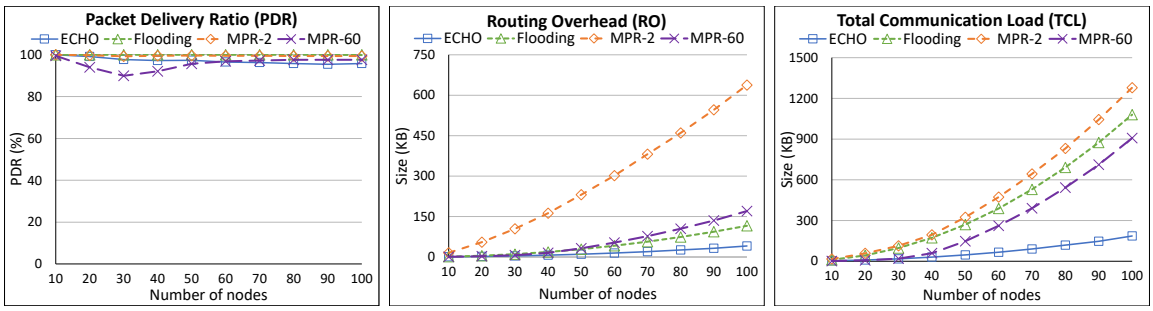
Figure 5.7: Simulation results for the increasing node speed scenario where the network size is 100 nodes but the node speed ranges from 4 to 20 m/s.

Even for large networks, having highly dynamic topology (node speed 20 m/s), ECHO provides ∼1.6x better PDR than MPR and ∼1.3x PDR than Flooding. ECHO incurs low interference and selects a strong backbone of source-independent critical nodes that ensures a high delivery ratio even in highly dynamic networks. Flooding's redundant transmissions cause high interference in large networks, resulting in low PDR. At high speeds, MPR-60 suffers from the infrequent update of source-dependent relay nodes, resulting in packet losses and low PDR. On the other hand, frequently updating the source-dependent relays increases the interference and packet collisions, resulting in low PDR for MPR-2.

Figure 5.7(b) shows ECHO having ∼2x lower RO than Flooding but almost the same as MPR at node speed 20 m/s. Flooding's TCL remains the same with increasing node speed, but MPR's TCL reduces with increasing node speed (shown in Figure 5.7(c)). ECHO's TCL is ∼4.5x lower than Flooding's TCL but higher than MPR's at high node speeds. MPR's lower TCL is due to its lower PDR.

5.5.5 Increasing Network Size (1 Mbps Data Rate)

This scenario repeats the simulation experiments of Section 5.5.1 with a 1 Mbps data rate. Figure 5.8(a) shows the PDR remaining almost the same for all protocols except for a dip observed for MPR for smaller networks (i.e., sizes between 20 and 40



(a) Packet Delivery Ratio (b) Routing Overhead (c) Total Communication Load

Figure 5.8: Simulation results for the increasing network size scenario where the size ranges from 10 to 100 nodes, the density remains constant, and the data rate is 1 Mbps.

nodes). The infrequent selection of relay nodes and an out-of-sync network topology results in the dip. However, as the network size increases, a higher node degree compensates for the infrequent selection by increasing the redundancy and decreasing the penalty of out-of-sync topology information. We have verified this phenomenon using a different set of randomly generated seeds and also a Hello interval of 30 seconds.

For networks of size 100, all protocols give almost 100% PDR, but ECHO requires up to 6.5x fewer transmissions than other protocols. Hence, the RO of ECHO is about 2.3x, 15.5x, and 4x lower than that of Flooding, MPR-2, and MPR-60, respectively (shown in Figure 5.8(b)). For the same reason, the TCL of ECHO is 5.8x, 6.8x, and 4.8x lower than that of Flooding, MPR-2, and MPR-60, respectively (shown in Figure 5.8(c)). We note that a lower RO and TCL will result in lower battery consumption.

The results in Figures 5.4, 5.5, 5.6, 5.7, and 5.8 show that ECHO is far more scalable than Flooding and MPR. The zero-control-packet nature of ECHO makes it a more practical solution, not only for wireless networks using low data rates but also for high data rates.

5.6 Conclusions

Network-wide broadcasting is a key requirement in most military multi-hop wireless networks. Most practically viable protocols utilize topology information to compute relay nodes. This information is typically collected using control packets that can be prohibitively expensive, especially in dense, low-bandwidth networks, and limits scalability. For example, we showed that the communication complexity of the well-known multi-point relay scheme is $O(N^2)$ in dense networks, as is the flooding scheme. Also, the 2-hop neighborhood information collected in the multi-point relay scheme may be more than required for broadcasting topology updates or any other information.

ECHO represents a radical departure from the prevalent thinking of collecting topology information via control packets to compute relay nodes. Rather, by using just two fields in the data packet header itself during occasional flooded packets, it learns source-independent critical nodes without control packets. Eliminating control packets makes the protocol more scalable, and invulnerable to control attacks. ECHO adapts to mobility, is tolerant of packet loss, and randomizes the set of critical nodes to balance battery consumption.

ECHO is simple to implement, robust, and scalable, making it a valuable protocol for real-world multi-hop wireless networks. While it is applicable to all multi-hop wireless networks, it is especially crucial for low-bandwidth, low-power applications such as short-burst long-range mobile networking for disaster relief, first responder networks, and Internet-of-Things (IoT), where the additional overhead of control packets or flooding is often unaffordable.

Simulation results have shown that ECHO significantly outperforms both Flooding and MPR with up to 20% improvement in the packet delivery ratio, while having up to 4x less total communication load. ECHO has a lower communication complexity ($O(N)$) in dense networks than both MPR [94] and Flooding ($O(N^2)$), while matching them for sparse networks. These improvements do not leverage any particular aspect of the MAC or RF, nor are dependent on the traffic in any particular way. Thus, ECHO

significantly enhances the scalability and lifetime of any multi-hop wireless network.

We have designed ECHO in collaboration with researchers at goTenna. It is implemented as a part of the goTenna Pro [4] – a small handheld device for the military, first-responders, and other professionals. It has significantly improved the performance of devices, resulting in their successful deployment in fighting forest fires, in the aftermath of hurricanes, and for military operations [19]. They pair with smartphones and function as Multi-hop Wireless Network (MWN) routers for forwarding messages, creating a mesh network with other such devices.

Chapter 6

VINE

In this chapter, we present a protocol, called VINE, designed for routing short-burst data in MANETs. Similar to the ECHO protocol described in Chapter 5, this protocol caters to the requirements of low-rate long-range networking discussed in Chapter 2.

Narrowband tactical communications (e.g., NATO NBWF [77]), off-grid disaster relief, long-range outdoor Internet-of-Things (IoT), and other contexts are characterized by low data rates. For example, the bit-rate for NBWF is 20-82 Kbps [18], and for the long-range IoT standard (LoRa) is 0.3-50 Kbps [10]. Further, these technologies are required to or are envisioned to operate in a mobile multi-hop context.

Routing protocols making use of dedicated control packets such as Hellos, Link-State Update, and Route Request/Response cannot support low-capacity MANETs even for modestly-sized networks. Section 6.4 shows AODV performing poorly for networks of size 30.

VINE is a novel routing protocol that does not utilize any routing control packets. Instead, VINE builds routing state by inspecting headers of data packets that it then uses for forwarding future data packets. Specifically, VINE uses three fields in the header, namely the sender, the previous sender, and a hop count to build routing state to 1-hop neighbors, 2-hop neighbors, and origin of the packet, respectively. Over time as traffic flows, an increasingly rich sink tree toward each node is created, resembling the growth of a "vine" in a "grove".

Nodes forward packets along non-increasing cost gradients (like water flowing downhill). If there is no fresh-enough gradient state, then the node broadcasts the

packet. This decision is taken independently at each hop – thus, a packet may alternate between broadcast (when no state exists) and unicast (when state exists) en route to its destination. Consequently, when the rate of topology change is so high that the routing state cannot keep up with it, VINE automatically leverages Flooding for ensuring a high delivery ratio. In addition to that, VINE provides per-hop reliability via *implicit acknowledgments*, that is, retransmissions based on overheard forwarded packets, and delivery notification via *end-to-end acknowledgments* – features that are not present in most routing protocols.

We have evaluated VINE theoretically and via ns3 simulations for a wide range of network sizes, densities, and traffic. We derive an expression for the communication complexity of VINE and show that it tends to stabilize quickly. We compare the performance of VINE with that of AODV because AODV is the basis of many standards, including RPL [117], LOADng [113], and IEEE 802.11s [63].

We first explain the current approaches used for routing packets in low-capacity networks. Then, we explain the VINE protocol and derive the communication complexity of VINE over time and analyze the trends. In the end, we present the results of simulation experiments for networks of varying size, density, load, and data rate.

6.1 Routing in Low-Power Wide Area Networks (LPWANs)

NBWF's evaluations in [77] identify all existing routing protocols to be inadequate for low-capacity networks and propose using link metrics in conjunction with the low-overhead Hello schemes in the original standard [18]. The works in [81] and [125] (called EP-RPL) describe the latest developments in LoRa [10] and RPL [117] technologies, respectively. A proof-of-concept implementation in [81] describes an extension to the single-hop LoRa networks. The work in [125] highlights the shortcomings of the traditional RPL protocols (e.g., (LOADng) [113] and P2P-RPL [26]) and describes an energy-efficient hybrid protocol that leverages the regional information in selecting a subset of nodes that participate in route discovery. However, both these

works are adaptations of AODV, applying route discovery procedures with extensive use of control packets.

VINE is based on a gradient-routing scheme, which has its roots in the directed diffusion approach described in [64], where the sink node floods control packets, acting as queries for collecting sensed data and building reverse paths to the sink. The reinforcement packets identify high-quality paths from the sources to the sink and deactivate low-quality ones. In one of the variants [55], the sink node selects "braided" (i.e., partially disjoint) paths from the sources to itself. In both these approaches, nodes know their next hop on the path to the sink node. Gradient broadcast approaches based on directed diffusion [100, 121] forward packets without knowing the next hop. Instead, a cost field in the header helps to forward packets in a non-increasing cost direction. The work described in [121] uses a "credit" scheme, in addition to the cost, for determining the "width" of the forwarding path (i.e., redundancy for delivering the packet to the sink).

In contrast to all the above approaches, VINE learns cost gradients by inspecting packet headers and uses them for routing future packets, eliminating the need for any control packets. It accommodates arbitrary source-sink pairs, and has a constant size header with only two extra fields compared to a typical stack, which we show to be insignificant compared to the control packet sizes. There is no periodic exchange of routing information, and hence, it incurs low overhead and little energy drain. Finally, the non-existence of explicit control packets or GPS information renders it immune to a wide-range of control and GPS attacks respectively.

6.2 The VINE Protocol

VINE is a network-layer protocol that efficiently delivers an application-layer message to the specified destination.

VINE learns routes by inspecting packet headers without using any control packets. It opportunistically sets up *gradient state* at each node determining the next hop and the cost for reaching a destination node. The header fields in every received

packet[1] are utilized by every node to create gradient state for the source (origin) of the packet, allowing the node to forward packets to the source along a reverse-path *sink-tree* rooted at the source. State is also created for nodes within a 2-hop neighborhood. The packet is forwarded along a non-increasing cost gradient (like water flowing downhill) to the destination. As traffic flows and every node originates packets, an increasingly rich set of sink trees form, resembling the growth of a "vine" in a "grove".

When gradient state for a destination is not available, or it is deemed outdated, the packet is *broadcast*, that is, all recipients are targets. When a state is available, the packet is *unicast*, that is, only the specified next hop processes the packet. This decision is taken independently at every hop. So the forwarding of a packet may alternate between flooding (in regions where no state exists) and unicast forwarding (in regions where state exists) en route to its destination. At network start-up time, when no state is available, the packets are flooded, but as traffic flows, nodes quickly create states and the need for flooding packets rapidly reduces (see Section 6.3). At the same time, under highly dynamic topology when nodes cannot keep up with the changes, VINE continues to provide a high delivery ratio by flooding packets.

Since many applications require a delivery notification, VINE incorporates an End-to-End Acknowledgment (E2E-A) scheme. Alternately, VINE could leverage a similar Transport Layer function. The E2E-A is handled like any other packet, and is also used to build gradients. In particular, the E2E-A allows a source to limit itself to a single flood per destination.

We first describe the procedure for building gradients, and then describe how the gradients are used for forwarding.

6.2.1 Gradient Establishment

A node maintains a list of gradients, one for each destination. Each entry includes the *destination*, *next hop*, *cost*, and *timestamp*. Currently, *cost* is the number of hops to the destination, but it can be generalized to any metric; *next hop* is the neighbor

[1] Going forward a packet refers to a data packet unless stated otherwise.

node to which the packet should be forwarded; and *timestamp* is the update time. If an entry does not get updated for a period (*GradientStateExpiry*), then it is deemed *expired* and not used. A single lowest cost entry is maintained for a given *destination* and *next hop* pair, but there can potentially be an entry for every *destination* and *next hop* pair. To limit gradient entries, each node maintains only a fixed number per destination (*MaxGradsPerDest*).

VINE packets contain the following header fields:

- *source*: The packet originator.
- *destination*: The packet destination.
- *sender*: The node forwarding the packet (same as the source if the sender is the originator).
- *prevSender*: The node that the sender first received the packet from (same as the source if sender is originator).
- *targetReceiver*: The intended next hop.
- *seqNum*: A sequence number unique at the source.
- *costFromSource*: the number of hops (cost) to the source.
- *ttl*: time-to-live (maximum forwarding count).

Algorithm 5 describes the steps followed when a reference node C receives a packet. The node updates (or creates) a 1-hop gradient towards the sender. In addition to that, the node creates a 2-hop gradient towards the previous sender with the sender as the next hop; and a k-hop gradient towards the source with the sender as the next hop, where k is the *costFromSource* field in the header. In the case of duplicate information (e.g., the source is the sender), only one gradient is created.

An unseen packet (i.e., not a part of the brief history of recently received packets, uniquely identified by its *seqNum* and *source*) is eligible for further processing. If the node itself is the destination, then the packet is sent to the application. If the receiving node is *targetReceiver* or the packet is broadcast, then the gradients are checked for packet forwarding. An entry for (packet, sender) is added to the history if not present.

Algorithm 5 VINE Gradient Establishment at node C

```
1: procedure PacketReceived(pkt)
2:     sndr ← pkt.sender; psndr ← pkt.prevSender
3:     src ← pkt.source; cst ← pkt.costFromSource
4:     trgt ← pkt.targetReceiver
5:     UpdateGradients (sndr, sndr, 1)                    ▷ Update grad to neighbor
6:     if src ≠ sndr and psndr ≠ C then
7:         UpdateGradients (psndr, sndr, 2)               ▷ Update grad to previous sender
8:     end if
9:     if src ≠ sndr and src ≠ psndr and psndr ≠ C then
10:        UpdateGradients (src, sndr, cst + 1)           ▷ Update grad to source
11:    end if
12:    if pkt ∉ pktHistory then                           ▷ Process if unseen
13:        if pkt.destination == C then
14:            Send pkt to application
15:        else if trgt == C or trgt == broadcast then
16:            ForwardPacket (pkt)                        ▷ Forward using Algorithm 6
17:        end if
18:    end if
19:    if (pkt, sndr) ∉ pktHistory then
20:        Add (pkt, sndr) to pktHistory                  ▷ Add to packet history
21:    end if
22: end procedure
23: procedure UpdateGradients(dst, nxtHop, cst)
24:    now ← currentTime
25:    if grads(dst) == null then                         ▷ If no grad available
26:        Add (dst, nxtHop, cst, now) to grads
27:    else
28:        g ← grads(dst, nxtHop)
29:        if g ≠ null and g.cost ≥ cst then              ▷ Grad has a higher cost
30:            g.cost ← cst
31:            g.timeStamp ← now
32:        else
33:            Add (dst, nxtHop, cst, now) to grads
34:            if |grads(dst)| > MaxGradsPerDest then
35:                Remove argmax_cost grads(dst)          ▷ Remove max cost grad
36:            end if
37:        end if
38:    end if
39: end procedure
```

6.2.2 Packet Forwarding

Algorithm 6 shows the steps followed for forwarding the packet. A node updates all necessary header fields before forwarding the packet. In particular, the node makes the sender as the previous sender, itself as the sender, and increments the cost by one. Then, the node searches the gradient table for a suitable next hop having the minimum cost to the destination with ties broken according to whether the next hop was already unsuccessfully used or *timestamp*. We note that this is but one of several possible heuristics for selecting the next hop. If no gradient entry is available for the destination, or all available entries are deemed expired based on the *timestamp*, then the node broadcasts the packet, else it uses a minimum cost gradient.

All VINE packets are sent using MAC-level broadcasts. Thus, with a WiFi MAC for example, no RTS, CTS or ACK packets are introduced, further eliminating control packets. Instead, VINE provides per-hop reliability based on overhearing the transmission of the next hop, considering it as an *implicit acknowledgment (IA)* and retransmitting if necessary[2]. A retransmission is scheduled using an IA timer only for the unicast packets.

The packet is retransmitted only if not in the history with the next hop as the sender. In the case of retransmission, the node attempts to select another min cost gradient, but if not available, then the same one is used, and the IA timer is reset. The *MaxRetransmissions* parameter determines the maximum attempts before the broadcast.

VINE also uses *end-to-end acknowledgments (E2E-A)* sent by the destination to the source, expressing a successful delivery of the packet. An E2E-A prevents nodes (e.g., last hop) from retransmitting in case of no IA. The E2E-A also allows the source and the intermediate nodes to learn gradients towards the destination. Nodes forward

[2] We recognize that implicit acknowledgments may not work in networks using directional transmission, but given the near-ubiquity of omnidirectional antennas and the significant control savings, we have decided to employ it.

Algorithm 6 VINE Packet Forwarding at node C

```
1: procedure ForwardPacket(pkt)
2:     Update necessary header fields in pkt
3:     if grads(pkt.destination) == null then
4:         pkt.targetReceiver ← broadcast                    ▷ No grad available
5:         Forward pkt
6:     else
7:         g ← argmin_cost grads(pkt.destination)
8:         trgt ← g.nextHop
9:         if (pkt, trgt) ∉ pktHistory then                  ▷ Forward if unseen
10:            pkt.targetReceiver ← trgt
11:            Forward pkt
12:            Schedule (IATimer, ReTransmit (pkt, {trgt}))  ▷ Schedule re-tx
13:        end if
14:    end if
15: end procedure
16: procedure ReTransmit(pkt, TrgtsUsd)
17:    if ∀ t ∈ TrgtsUsd, (pkt, t) ∉ pktHistory then        ▷ Forward if no IA
18:        if |TrgtsUsd| ≤ MaxRetransmissions then          ▷ Check tx attempts
19:            dst ← pkt.dest
20:            trgt ← pkt.targetReceiver
21:            g ← argmin_cost grads(dst, trgt ∉ TrgtsUsd)  ▷ Get unused target
22:            if g ≠ null then
23:                pkt.targetReceiver ← g.nextHop
24:            end if
25:            Add pkt.targetReceiver to TrgtsUsd
26:            Schedule (IATimer, ReTransmit (pkt, TrgtsUsd))  ▷ Schedule re-tx
27:        else
28:            pkt.targetReceiver ← broadcast
29:        end if
30:        Retransmit pkt
31:    end if
32: end procedure
```

an E2E-A in the same way as any other data packet. Nodes that have received an E2E-A for a packet P do not forward P.

6.2.3 Discussion: Flooding and Control Information

The use of flooding as an initial or default mode may raise concerns about an excessive load. However, VINE achieves a natural balance, flooding a packet only when

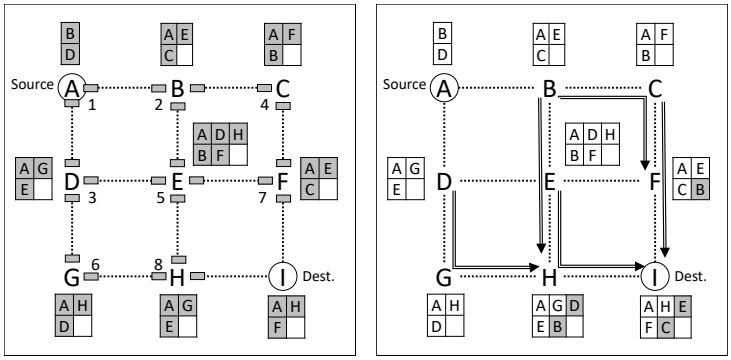
there is no or expired gradient, but that implies no or low traffic, so flooding can be accommodated. On the other hand, when traffic is high, nodes maintain up-to-date gradients, reducing the need for flooding. We show in Section 6.3 that the load quickly stabilizes as a result of gradients learned from the previous sender information, IA, and E2E-A. Further, we note that traditional routing protocols also use flooding for disseminating control packets.

It may be argued that while VINE does not have dedicated control packets, there is control information in the header. However, we note that all of the fields except *prevSender* and *costFromSource* are present at some layer of any MANET stack – for instance, the MAC header typically contains the *sender* and the *targetReciever*. Further, *costFromSource* is strictly not necessary as it can be derived as (MAX_TTL - *ttl*) where MAX_TTL is initial value of *ttl*. Thus, the total impact is just the length of *prevSender*. Moreover, it does not incur the per-packet MAC- and PHY-layer header and MAC contention penalties that dedicated additional control packets do. These points are confirmed by the simulation results in Section 6.4.

6.2.4 VINE Example

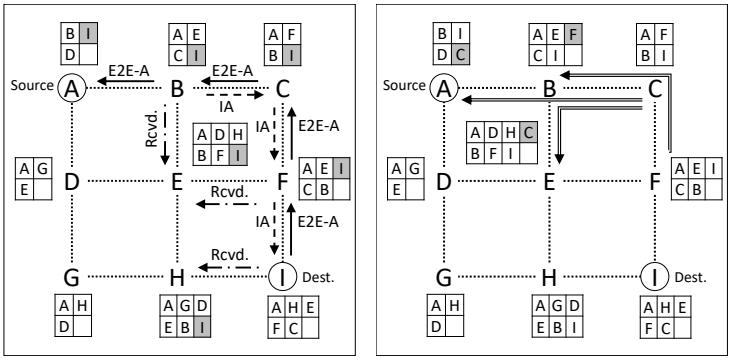
Figure 6.1 shows an example VINE operation, in which node A generates a packet for node I, but there is no gradient at node A, so the packet is flooded. Figure 6.1(a) shows a possible transmission order based on a random jitter. The square boxes next to the nodes indicate the gradients learned towards the source and the sender after the flooding completes. Figure 6.1(b) shows the gradients (shaded) learned towards the previous sender, depending on the transmission order. For example, node F could have had a gradient towards node D, if node D would have transmitted the packet before node B did.

Figure 6.1(c) shows the E2E-A sent by node I to node A and the gradients (shaded) learned towards the source and the sender of the E2E-A. Since all nodes have gradients towards node A, the E2E-A is unicast to the next hop, setting an IA timer for retransmissions. In addition to the next hops, a few other nodes (e.g., nodes H



(a) States learned towards the source and the sender

(b) States learned towards the previous sender



(c) States learned towards the source and the sender of the E2E-A

(d) States learned towards the previous sender of the E2E-A

Figure 6.1: An example VINE operation, in which node A sends a packet to node I. Several nodes learn gradients towards the source, the sender, and the previous sender of the packet as well its E2E-A. Shaded boxes indicate new states learned, and unshaded boxes indicate states carried over from the previous step.

and E) also receive the E2E-A and learn gradients towards the source and the sender. Node A gets the E2E-A for its data packet, but an IA timer is not set for the E2E-A when the destination is the next hop (i.e., node A). Figure 6.1(d) shows the gradients learned towards the previous sender of the E2E-A. This example shows nodes learning more than 50% of the gradients (on average) from just a single packet and its E2E-A.

6.3 Communication Complexity Analysis

We now analyze the communication complexity (CC) of VINE. We define CC as the total bytes transmitted in the network for a packet originated at source X.

Table 6.1: Symbols

Category	Symbol	Meaning
Network	**N**	Network Size
	D	Network Diameter
	M	Average Node Degree
	B	Data Packet Size
	R_{gen}	Data Packet Generation Rate
Messages	b	VINE Header Size
	E	E2E-A Size
Protocol	**T**	Gradient State Expiry Time

Table 5.1 shows all symbols used in the CC analysis. Let N and D be the network size and the network diameter, respectively. The message generated (payload) at the node is considered to be of size B bytes. The VINE header added to the packet is of size b bytes. The E2E-A acknowledgment packet sent by destination node Y is of size E bytes.

The packet originated at node X is unicast to the next hop, if node X has a valid gradient to node Y; otherwise, it is broadcast. Node X has a valid gradient to node Y if it sent a packet to node Y in the last T seconds and received its corresponding E2E-A. Here, T is the *GradientStateExpiry* period. We consider all transmissions to be error-free.

We assume that node X generates packets for the destinations selected uniformly at random. Thus, the probability of not generating a packet for a particular destination is $(N-2)/(N-1)$.

Let R be the packet generation rate. If node Y is the destination of a packet generated at node X, the probability of broadcast is at most the probability of not

sending any of the previous $R_{gen}T$ packets to node D. We use the term "at most" because a route to node Y can be learned using other means like IA and previous sender.

Thus, the probability of sending n^{th} packet as a broadcast is:

$$p_n \leq \left(\frac{N-2}{N-1}\right)^{min(R_{gen}T, n-1)} \tag{6.1}$$

The exponent uses min to capture the fact that if T is really high, say infinity, then the probability of broadcast depends on the previous $n-1$ packets, else it depends on the previous $R_{gen}T$ packets. None of these previous $R_{gen}T$ packets must be sent to node Y for the n^{th} packet to be broadcast. From Equation 6.1, as $R_{gen}T \to \infty$, $p_n \to 0$. However, for practical values of $R_{gen}T$, the probability of broadcast will be infinitesimally small.

$$CC_n \leq p_n[(N-1)(B+b) + dE] + (1-p_n)[d(B+b) + dE] \tag{6.2}$$

The first term of Equation 6.2 represents the worst case complexity of broadcast. Here we assume that the packet is flooded. When the packet reaches node Y, all the intermediate nodes and node Y will have routes to node X. So E2E-A will be forwarded maximum D times (since D is the network diameter).

The second term represents the complexity when node X has a valid gradient to node Y. If node X has a valid gradient, then all the intermediate nodes between X and Y would also have valid gradients because the E2E-A of the previous packet would have updated them. Thus, the packet from node X will be forwarded maximum D times, and so will be the E2E-A.

We compare the average CC obtained from Equation 6.2 to the average CC of an ns3 experiment, in which a single node sends packets to destinations selected uniformly at random in a 6x6 Manhattan grid. The experiment uses wireless links working in ad hoc mode and susceptible to interference and packet collisions. So, unlike the theoretical analysis, the nodes may retransmit packets based on the explanation in

Section 6.2. The average CC from Equation 6.2 (Numerical) is calculated using the ns3 experiment values (e.g., network size, payload). A random variable selects the destination and accordingly determines whether the node unicasts or broadcasts the packet.

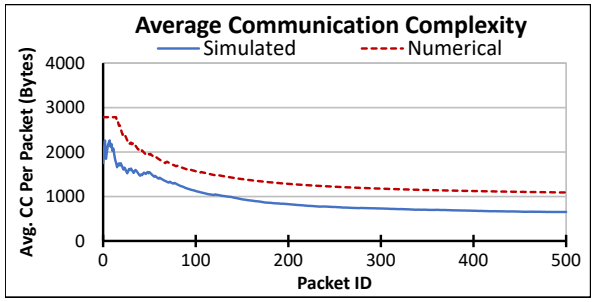


Figure 6.2: Average communication complexity

Figure 6.2 shows the average CC of the first 500 packets. The simulated CC is lower than the numerical CC because, in addition to the source and the sender, nodes also learn gradients towards the previous sender, reducing the need for flooding. The average CC is high for the first few packets due to initial flooding, but it quickly converges as nodes start forwarding packets as unicast. In this experiment, only a single node generates packets, but when all nodes start generating, then the average CC would converge quicker.

One could argue that a certain amount of traffic churn is required to maintain state. Figure 6.3 shows that in a 6x6 grid if a single node is sending data packets to random destinations and $T = 1$ min, then the "sweet spot" is very low, i.e., VINE attains lower CC than Flooding for $R_{gen} >= 4$ packets/min.

6.4 Simulation Results

We have implemented and evaluated VINE in ns3 and compared the results to that of AODV. Table 6.2 lists the scenarios used for evaluation. In these scenarios, nodes transmit at 25 Kbps data rate to model low-capacity networks. We have also

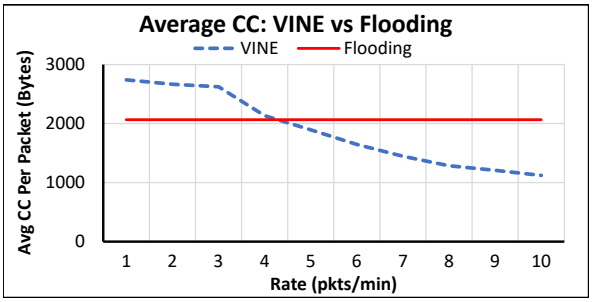


Figure 6.3: Churn analysis showing the "sweet spot" between VINE and Flooding.

evaluated VINE at a higher data rate by repeating the increasing size experiments and configuring nodes to transmit at 1 Mbps data rate. Table 6.3 lists all the simulation parameters.

Table 6.2: Simulation Scenarios

Simulation Scenario	**Network Size** (nodes)	**Network Density** (nodes/km^2)	**Data Packet Interval** (seconds)	**Data Rate**
Increasing Network Size	[10, 50]	3.3	30	25 Kbps
Increasing Network Density	30	[0.83, 7.5]	30	25 Kbps
Increasing Network Load	30	3.3	[10, 30]	25 Kbps
Increasing Network Size	[10, 50]	3.3	30	1 Mbps

AODV simulations are performed using two different Hello intervals, namely 1 and 30 seconds, and their results are shown in the graph plots as AODV-default and AODV-modified, respectively. The recommended Hello interval in the AODV RFC [90] and ns3 is 1 second. However, we have found that the performance of AODV improves in our settings if we use large Hello intervals and other associated parameters. This set of parameters providing improved AODV performance is shown under the "AODV-modified" column of Table 6.3. Hereafter, the RFC default and improved versions of AODV will be referred to as AODV-default and AODV-modified, respectively.

Table 6.3: Simulation Parameters

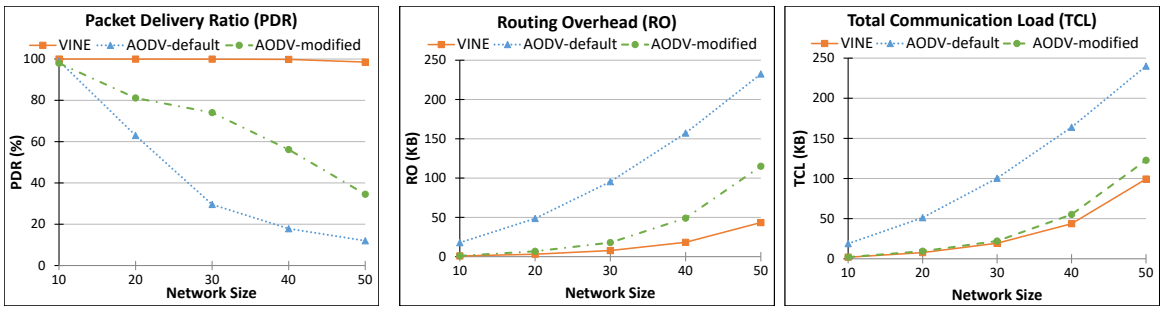
Parameter	**Value**	**Parameter**	**Value**
Simulation Time	60 minutes	Node Speed	4 m/s
Data Packet Size	50 Bytes	Node Mobility	Rand. Waypoint
Propagation Loss	Friis Model	MAC	802.11b
VINE			
GradientStateExpiry	60 secs	IATimer	0.5 secs
MaxRetransmissions	2	MaxGradsPerDest	2 secs
AODV-default		**AODV-modified**	
Hello Interval	1 sec	Hello Interval	30 secs
Node Traversal Time	40 ms	Node Traversal	0.25 secs
Next Hop Wait	50 ms	Next Hop Wait	0.25 secs
Active Route	3 secs	Active Route	90 secs
MyRoute Timeout	6 secs	MyRoute Timeout	180 secs

The simulation results are compared using the following two metrics: (1) Packet Delivery Ratio (PDR) is the ratio of the total data packets received and transmitted, (2) Routing Overhead (RO), which is the total size of the packet headers, E2E-A (in VINE), and control packets (in AODV), transmitted per minute, and (3) Total Communication Load (TCL) is the total bytes transmitted per minute, comprising of the data packet headers, E2E-A (in VINE), control packets (in AODV), and the payload sizes.

All numerical comparisons between VINE and AODV are with the better-performing AODV-modified unless mentioned otherwise.

6.4.1 Increasing Network Size

Figure 6.4(a) shows VINE having a significantly higher PDR than AODV for all network sizes. In fact, the PDR is close to 100% for all network sizes and ∼2.5x better than AODV for size 50 nodes. AODV's PDR is remarkably low at all but very small network sizes showing its inadequacy over low data rates. Nodes transmitting at low data rates experience long transmission delays, making the network susceptible



(a) Packet Delivery Ratio (b) Routing Overhead (c) Total Communication Load

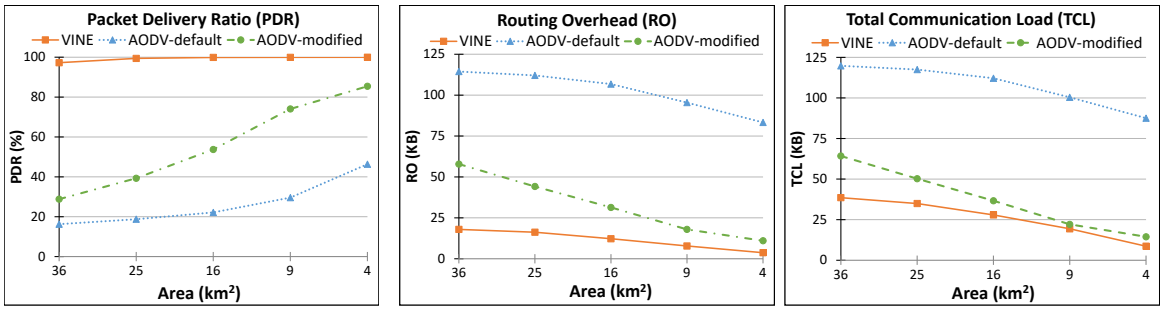
Figure 6.4: Simulation results for the increasing network size scenario where the size ranges from 10 to 50 nodes but the density remains constant.

to packet losses due to increased congestion, interference, and collision. AODV suffers the most due to its extensive use of route discovery, route maintenance, and local connectivity procedures. The route discovery procedure of AODV requires flooding every Route Request (RREQ) message, followed by an exchange of Route Reply (RREP) and Route Reply Acknowledgment (RREP-ACK) messages. The local connectivity procedure sends Hello messages periodically for discovering and maintaining links to neighbors. Every packet loss is assumed to be caused by a link failure, triggering the route maintenance procedure of sending Route Error (RERR) messages and following it with the route discovery procedure. The network that was already congested and experienced a packet loss becomes overwhelmed with the control packets, resulting in a considerably low overall PDR. VINE uses data packets for learning routes, so the absence of control packets reduces the possibility of collision. Although a route's absence or expiry results in broadcast, the resulting flooding updates routes in several nodes. A packet is also broadcast in the absence of an IA, but only after maximum retransmission attempts. So not every packet loss results in a broadcast. The E2E-A feature unique to VINE also helps update routes frequently, making VINE resilient against dynamic topology changes and achieve a high PDR.

Despite flooding packets when necessary and using E2E-As, the RO in VINE is ∼2x better (lower) than in AODV (shown in Figure 6.4(b)), proving that the overhead

of extra fields in the header is insignificant compared to the overhead of control packets. Figure 6.4(c) shows AODV overwhelming the network with control packets because its RO and TCL are almost the same, yet VINE's TCL is up to 1.2x less than that of AODV.

6.4.2 Increasing Network Density



(a) Packet Delivery Ratio (b) Routing Overhead (c) Total Communication Load

Figure 6.5: Simulation results for the increasing density scenario where the network size is 30 nodes but density ranges from 0.83 to 7.5 nodes/km^2 (i.e., simulation area ranges from 36 to 4 km^2).

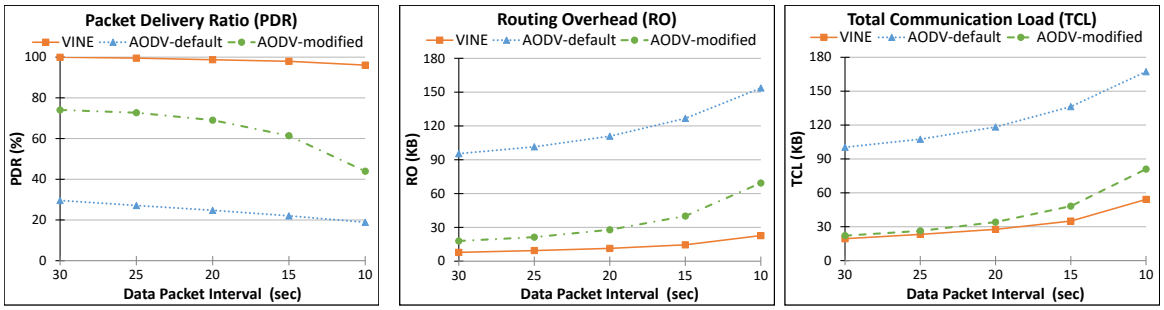
Figure 6.5(a) shows VINE having ∼3x and ∼1.2x better PDR than AODV in sparse (36 km^2) and dense (4 km^2) networks, respectively. In dense networks, the nodes are in close proximity to each other and frequently update their routes. The absence of control packets in VINE keeps the interference low. Sparse networks are more susceptible to link breaks, but VINE benefits from network layer retransmission, either to a different neighbor or broadcast, ensuring resiliency and a high PDR. Moreover, the previous sender information, IA, and E2E-A help frequently update the routes and minimize the need for flooding packets.

AODV's low PDR in sparse networks is due to two reasons. First, unlike VINE, there is no network layer retransmission to another neighbor, so a transmission over a broken link always results in a packet loss. Secondly, every packet loss triggers the route maintenance procedure, followed by the route discovery procedure, increasing interference for other ongoing transmissions. Although the local connectivity procedure

detects link breaks, it takes two failed Hello messages to confirm one [90]. The PDR improves in the denser networks because the network experiences fewer link breaks, and hence, less frequent route discovery and route maintenance instantiations.

Figure 6.5(b) shows RO reducing for AODV and VINE with increasing density because fewer link breaks reduce the route discovery and route maintenance procedure calls (in AODV) and floodings (in VINE). In sparse networks, VINE has ∼3x better (lower) RO than AODV, but the gap narrows in dense networks. The TCL also reduces for both AODV and VINE, as shown in Figure 6.5(c), because packets get forwarded by fewer nodes in dense networks.

6.4.3 Increasing Network Load



(a) Packet Delivery Ratio (b) Routing Overhead (c) Total Communication Load

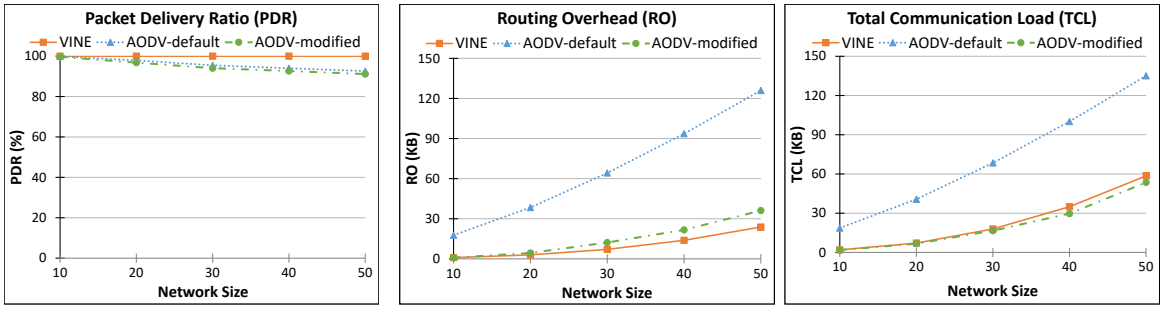
Figure 6.6: Simulation results for the increasing network load scenario where the network size is 30 nodes but packet interval ranges from 10 to 30 secs.

Increasing the network load (i.e., decreasing the data packet interval) is likely to increase packet losses due to growing interference and collisions. However, VINE's zero-control characteristic helps achieve up to 2x better PDR than AODV (shown in Figure 6.6(a)). On the other hand, AODV experiences a steep drop in the PDR because of its vicious circle, in which every packet loss due to a collision triggers the route maintenance and route discovery procedures, increasing the control packets and interference even further. Figure 6.6(b) shows RO increasing in AODV despite the decreasing PDR and the same number of Hello messages, proving that the route

maintenance and route discovery procedures are triggered frequently. The RO increases in VINE as well, but that is because more packets are forwarded at smaller intervals, yet VINE has ∼2x better (lower) RO than AODV.

Increasing RO is also reflected in the TCL in Figure 6.6(c), where a majority of the TCL in AODV is due to the control packets because the PDR is low and most of the packets fail to reach their destination. VINE delivers most of the packets even at high network load and ensures a high PDR, and hence, experiences an increase in the TCL.

6.4.4 Increasing Network Size (1 Mbps Data Rate)



(a) Packet Delivery Ratio (b) Routing Overhead (c) Total Communication Load

Figure 6.7: Simulation results for the increasing network size where nodes are configured to transmit at 1 Mbps data rate.

We now investigate if VINE's significant performance advantage extends to higher data rates, as might be expected in WiFi-based networks, by repeating the increasing network size experiments with a data rate of 1 Mbps. Figure 6.7(a) shows that both AODV and VINE have high PDRs, but VINE has 10% better PDR than AODV. The high PDR indicates that there is little interference at high data rates and the given network loads. Similar to the previous results, VINE has nearly 100% PDR for all network sizes, whereas AODV's PDR reduces marginally in large networks. The fact that AODV-default and AODV-modified have similar PDRs proves that the 30

seconds Hello interval is suitable for all considered network scenarios and that AODV-modified provides a fair comparison to VINE.

Despite having similar PDRs, there is a significant difference between the ROs of AODV-default and AODV-modified (shown in Figure 6.7(b)). The difference is mainly due to 30x more Hello messages sent by AODV-default. However, the RO in VINE is ∼1.2x (for size 50 nodes) better than AODV-modified, mainly due to the large size control packets of AODV. On the other hand, the TCL in AODV-modified is similar to the TCL in VINE because VINE leverages flooding for ensuring a high PDR at the expense of a few extra packets forwarded. The overhead of these extra packets is reflected in Figure 6.7(c), but it is significantly lower compared to that of AODV-default.

VINE outperforms AODV in all scenarios, ensuring versatility, reliability, and resilience against dynamic topology changes, proving that it is a better protocol, in general, for all network types and scenarios.

6.5 Conclusions

Existing routing protocols such as AODV and OLSR for multi-hop wireless networks are universally based on disseminating explicit control packets. In low-capacity low-traffic MANETs such as those based on NBWF [18] and LoRa [10], the use of control-packet-based protocols such as AODV causes premature congestion collapse as described in Section 6.4.

We have presented a novel routing protocol called VINE that does not use dedicated control packets. Instead, routes are progressively built and refined based on observations of data packets. Our study shows that while the first few packets are flooded, VINE rapidly builds sufficient state to stabilize the communication complexity. Our simulation results across a wide range of network size, density, traffic load over low capacity networks show that VINE is substantially more reliable than AODV (e.g. PDR of 98% vs AODV's 42% for 50 nodes). The total communication load, which is a rough measure of battery energy consumption, is also lower for VINE.

Similar to the ECHO protocol, we have designed VINE in collaboration with the researchers at goTenna. The goTenna Pro mesh networking device – designed for emergency, public safety, and military – uses VINE for applications such as 1-1 texting, unicasting low-fidelity images, and other low-bandwidth delay-tolerant and short-burst applications. However, VINE applicability extends more generally to any MANET as a reliable and scalable protocol.

Chapter 7

CENTRALIZED OPPORTUNISTIC REACTIVE ROUTING

In this chapter, we present a routing protocol, called Centralized Opportunistic Reactive Routing (CORR), designed for the SD-MANET architecture described in Chapter 3. The CORR protocol addresses the challenges faced by the PCC protocol discussed in Section 4.4; in particular, the issues concerning the communication overhead and unreliable transmission of control messages.

The results of the PCC protocol described in Section 4.3 indicate that learning the network topology at a centralized location (i.e., the SDNC) using the neighbor information of all nodes results in the transmission of several control messages (i.e., high communication overhead). Moreover, sending route updates to all nodes further increases the overhead and causes congestion at the SDNC. As a result, several neighbor information messages get dropped, especially because they are not sent using reliable schemes, leading to SDNC learning incomplete or disconnected network topology.

CORR addresses the above issues using the approaches discussed in the ECHO and VINE protocols. CORR uses the ECHO protocol to identify a subset of nodes (i.e., the critical nodes) for learning the network topology. The SDNC sends the routing information only to these critical nodes, making them the network backbone for forwarding data packets. ECHO selects critical nodes that form a Connected Dominating Set (CDS) of the network graph. The fact that critical nodes form a CDS allows each node to have a connected path to every other node in the network.

The Implicit Acknowledgment (IA) scheme discussed in Chapter 6 allows VINE to improve the per-hop reliability of data packets. CORR employs this IA scheme for sending the NI messages and improving its per-hop reliability. As a result, the

likelihood of SDNC receiving all the transmitted NI messages and learning a connected network topology increases.

Unlike PCC, CORR is designed to be a reactive protocol, in which the SDNC learns the network topology *proactively* but sends the routing information *reactively*. The SDNC floods a message periodically that allows nodes to maintain their route to SDNC as well as identify critical nodes in the network. *Only* critical nodes send their neighbor information in the NI messages, allowing SDNC to maintain up-to-date network topology. However, the SDNC sends the routing information only on receiving requests from nodes.

On receiving a request, the SDNC selects routes to the destination in the request message for *all* critical nodes, irrespective of the request originator, and disseminates the routing information via network-wide broadcasts. All nodes *opportunistically* update their forwarding tables. Thus, a single route request results in updating the forwarding tables of all nodes, suppressing the need for initiating multiple route requests for a particular destination. As traffic flows and the SDNC receives several route requests, it rapidly sends routes updates for all destinations and creates a network backbone for forwarding data packets.

We first describe the CORR protocol and then explain its communication complexity. In the end, we explain the results of its extensive evaluation.

7.1 The CORR Protocol

Similar to PCC, we describe CORR using the following three functions: (1) learning route to SDNC, (2) learning network topology, and (3) sending network routes. The three managers inside the SDNC perform the three functions.

7.1.1 Learning Route to SDNC

The SDNC periodically floods a message called Topology Discovery (TD), containing fields for *seqNum* and *prevSender*. The *seqNum* field allows identifying new TDs. The *prevSender* field allows using the ECHO protocol for selecting the critical

nodes. Similar to the PCC protocol, nodes retransmit the new TDs, and in this process, learn their RTS and update their neighbor list. In addition to performing these steps, they also update the *prevSender* field. The sender of the TD message (i.e., a neighbor) becomes the previous sender.

Note that, in ECHO, the data packet header includes the *prevSender* field, whereas, in CORR, the TD message (a control packet) includes it. In ECHO, the node that initiates the Full Flood (FF) procedure becomes critical. In CORR, only the SDNC initiates TD flooding, so it is always critical.

At the end of the TD flooding, all nodes identify their state (i.e., critical or non-critical), recognize their neighbors, and learn their route to the SDNC (RTS). Figure 7.1 shows a network topology with one possible set of critical nodes.

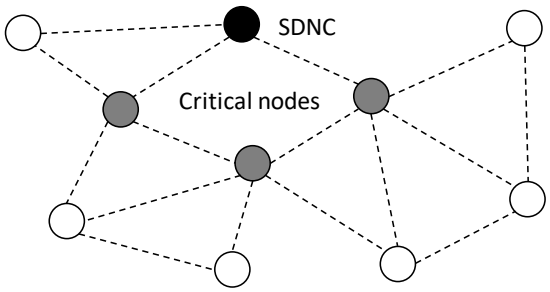


Figure 7.1: A network with SDNC and one possible set of critical nodes (shaded).

Algorithm 7 describes the procedures for learning the route to SDNC. The SDNC periodically calls *SendTD* every *TDInterval*. In this procedure, SDNC increments the previously sent sequence number and assigns it to the *seqNum* field, sets the *prevSender* field to itself, and broadcasts the message. The nodes that receive the message call *ProcessTD* and include the sender in their neighbor list. If the TD is new, nodes set their state to *pending*, record the *seqNum*, and update their RTS. They also update the *prevSender* field before retransmitting the message. In the end, the node schedules a call to *EchoTimerExpired* with a *RoundTripTime (RTT)* delay because a node meant to be critical will get an echo within that period. When a pending node receives an

echo, it sets itself to critical and schedules a call to *SendNI* using an *RTT* delay. During this period, the node expects to receive TDs from all its neighbors and update its neighbor list. In the call to *EchoTimerExpired*, if the state is *pending*, it is set to *non-critical*.

Algorithm 7 Learning Route to SDNC

```
1: procedure SendTD
2:     TD.seqNum ← TDSeqNum++
3:     TD.prevSender ← SDNC
4:     Broadcast TD
5:     Schedule (SendTD, TDInterval)
6: end procedure
7: procedure ProcessTD(TD)
8:     neighbors.insert (TD.sender)
9:     if TD.seqNum > savedSeqNum then
10:        state ← pending
11:        savedSeqNum ← TD.seqNum
12:        RTS ← TD.sender
13:        TD.prevSender ← TD.sender                ▷ Update previous sender
14:        Broadcast TD
15:        Schedule (EchoTimerExpired, RTT)
16:    else if TD.prevSender is itself and state is Pending then      ▷ Echo received
17:        state ← critical
18:        Schedule (SendNI, RTT)                   ▷ Call to Algorithm 8
19:    end if
20: end procedure
21: procedure EchoTimerExpired
22:    if state is Pending then
23:        state ← non-critical                     ▷ No echo
24:    end if
25: end procedure
```

7.1.2 Learning Network Topology

Only critical nodes send their neighbor information to SDNC in the NI messages. Nodes forward the messages via their RTS. The SDNC collects all these messages and learns the network topology. Here, CORR follows the ubiquitous assumption of bidirectionality. However, the learned network topology is expected to have missing links

Algorithm 8 Learning Network Topology

```
1: procedure SendNI
2:     NI.neighbors ← neighbors
3:     NI.source ← itself
4:     NI.seqNum ← savedSeqNum
5:     NI.nextHop ← RTS                                          ▷ Send to RTS
6:     DeliveryAttempts ← 1
7:     Unicast NI
8:     if RTS ≠ SDNC then
9:         Schedule (IATimer, ReTransmit (NI))                   ▷ Schedule re-tx of NI
10:    end if
11: end procedure
12: procedure ReTransmit(NI)
13:    if NI ∉ NIHistory then                                    ▷ If no IA
14:        if DeliveryAttempts ≤ MaxRetransmissions then         ▷ Check re-tx attempts
15:            DeliveryAttempts++
16:            Retransmit NI
17:            Schedule (IATimer, ReTransmit (NI))               ▷ Schedule re-tx of NI
18:        else
19:            NI.nextHop ← Broadcast
20:            Broadcast NI
21:        end if
22:    end if
23: end procedure
24: procedure ProcessNI(NI)
25:    if NI ∉ NIHistory then                                    ▷ If unseen NI
26:        Add NI to NIHistory
27:        if state == critical then
28:            if NI.nextHop == itself OR NI.nextHop == Broadcast then
29:                NI.nextHop ← RTS
30:                DeliveryAttempts ← 1
31:                Unicast NI
32:                if RTS ≠ SDNC then
33:                    Schedule (IATimer, ReTransmit (NI))       ▷ Schedule re-tx of NI
34:                end if
35:            end if
36:        end if
37:    end if
38: end procedure
```

(i.e., between the non-critical nodes). But only critical nodes forward data packets, so the missing links between non-critical nodes are not required. Moreover, each node,

including a non-critical node, is aware of its neighbors (from the TD flooding) and forwards data packets directly to them if they are the destination.

Algorithm 8 describes the procedures used for learning the network topology. In the call to *SendNI*, the node prepares the NI message, sets the *neighbors* and the *seqNum* fields, and unicasts the NI message to its RTS. The node sets *DeliveryAttempts* to 1 and schedules a call to *ReTransmit* with *IATimer* as the delay.

The nodes that receive the NI message call the *ProcessNI* method. In this method, the node checks if the received message is a part of *NIHistory* (i.e., a brief history of recently received NIs, uniquely identified by its *seqNum* and *source*). If not, then the received NI is included in the *NIHistory*. If the receiving node is the next hop in the message or NI is broadcast, then the node forwards the message to its RTS and schedules a call to *ReTransmit* with *IATimer* as the delay. When the sender senses the transmission of next hop (i.e., implicit acknowledgment), it calls *ProcessNI* and adds the sensed NI message to its NIHistory to prevent itself from retransmitting.

In the *ReTransmit* method, if NI is not a part of the *NIHistory*, then the node retransmits. During this retransmission, if $DeliveryAttempts \leq MaxRetransmissions$, the node unicasts to the next hop; otherwise, it broadcasts.

The Implicit Acknowledgment (IA) scheme used for sending the NI messages is similar to the VINE's IA scheme described in Chapter 6. The IA scheme in CORR improves the per-hop reliability for NI messages and increases the likelihood of SDNC learning a connected network topology.

7.1.3 Sending Network Routes

The SDNC selects critical nodes and learns network topology periodically but sends network routes *only* on receiving requests from nodes. When nodes do have routes for forwarding data packets, they broadcast. Only critical nodes forward, either as unicast or broadcast, depending on their route availabilities. The CDS property ensures successful deliveries of data packets to their destination.

7.1.3.1 Sending RU Messages

In the absence of a route, when a critical node broadcasts the data packet, it sends a message called Route Request (RR) to SDNC via the RTS. This message includes the destination to which the route was not available. On receiving, the SDNC selects routes to the *destination* for *all* critical nodes, irrespective of the request originator. Then the SDNC includes the selected routes and a sequence number in a message called Route Update (RU) and broadcasts it. Each critical node rebroadcasts, just once, by checking the sequence number. Note that this is network-wide broadcast from the SDNC for disseminating the routing information. So, all critical nodes receive the message. All non-critical nodes also receive as per the properties of CDS. Critical nodes update their forwarding table using the routing information in the message. Non-critical nodes also update their forwarding table using the routing information of critical nodes. For a non-critical node, a critical (neighbor) node having the shortest path to the destination becomes the next hop.

7.1.3.2 Forwarding Data Packets

As traffic flows and critical nodes send RRs for several destinations, the SDNC rapidly populates the forwarding tables of all nodes. As a result, a strong network backbone gets created for forwarding the data packets, suppressing the need for broadcasting subsequent data packets and sending multiple RRs for the same destination.

Nodes invalidate their configured routes on detecting link breaks. A node detects a link break towards a neighbor (i.e., the next hop) when the node fails to receive a packet from that neighbor in the past *NbrMaintenance* interval. Nodes may also detect link breaks using the 802.11 CSMA/CA schemes (i.e., RTS/CTS/ACK), if available. In the case of an invalid route, the node broadcasts the data packet. Critical nodes that receive the broadcast packet check their forwarding table for valid routes. If available, then the node unicasts to the next hop, else it broadcasts. Thus, a data packet may alternate between broadcast (i.e., from nodes with no or invalid routes) and unicast (i.e., from nodes with valid routes) en route to its destination. However, in

Algorithm 9 Sending Network Routes And Forwarding Data Packets

```
1: procedure ForwardDataPacket(pkt)
2:     if state == critical or node == pkt.source then
3:         if route to pkt.dst is valid then
4:             Unicast pkt to the nextHop
5:         else if state == non-critical and RTS is valid then
6:             Unicast pkt to RTS                    ▷ Forward to RTS if no valid route
7:         else
8:             Broadcast pkt
9:             if state == Critical and node ≠ SDNC then
10:                if RR for pkt.dst not seen in past NTT then
11:                    RR.dst ← pkt.dst
12:                    RR.seqNum ← RRSeqNum++
13:                    Unicast RR to RTS                ▷ Send route request
14:                end if
15:            else if node is SDNC then
16:                SendingNetworkRoutes (pkt.dst)
17:            end if
18:        end if
19:    end if
20: end procedure
21: procedure SendingNetworkRoutes(dst)
22:    if routes to dst not selected in past 3*NTT then
23:        for each critical node c do          ▷ Select routes for all critical nodes
24:            r ← route between c and dst
25:            Include r in RU
26:        end for
27:        RU.seqNum ← RUSeqNum++
28:        Broadcast RU                  ▷ Network-wide broadcast via critical nodes
29:    end if
30: end procedure
```

highly dynamic networks, when nodes experience frequent link breaks, and the SDNC fails to keep up, then critical nodes ensure successful deliveries via the network-wide broadcasts.

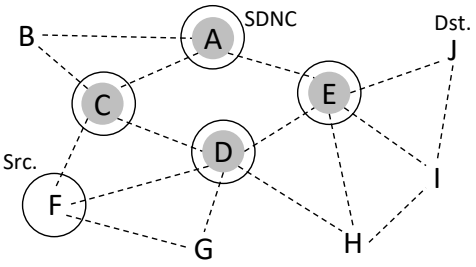
Algorithm 9 describes the procedures for sending network routes and forwarding data packets. When a node receives a data packet, either from the application or another node, it follows the following steps. If the node is critical, or it is the packet source, then the packet is forwarded. If a valid route to the destination is available,

then the packet is unicast to the next hop. If a non-critical node (i.e., the packet source) with no route has a valid RTS, then the packet is unicast to the RTS. Unicasting to the RTS prevents multiple critical nodes from originating the RR messages for the same destination. Nodes broadcast the data packet in all other scenarios. Nodes in the *pending* state always broadcast.

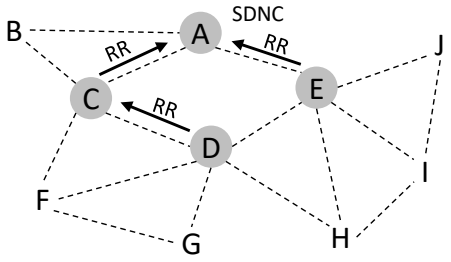
When a critical node broadcasts, it also sends an RR message to the SDNC if it has not seen one for the same destination in the past *NetworkTraversalTime (NTT)* seconds. This interval is used to avoid duplicate RR transmissions and originations.

The SDNC is also a critical node and broadcasts the data packet in the absence of a route, but instead of sending an RR message, it calls the SelectRoutes procedure. In this procedure, the SDNC selects routes to the destination if it has not already done so in the past $3 * NTT$ period – thus, avoiding redundant RU transmissions. The SDNC selects routes for each critical node, includes them in an RU, and broadcasts, expecting it to be a network-wide broadcast and all nodes getting the message. All nodes update their forwarding tables with the routes in the RU.

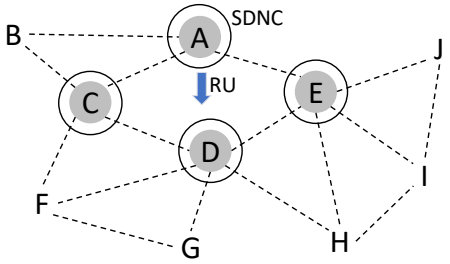
Figure 7.2 illustrates an example of data packet forwarding. Node F has a data packet to send to node J. The SDNC has already identified the critical nodes (shown as shaded) and learned the network topology. Nodes have routes towards their neighbors (learned during TD flooding) but not towards any other node. Therefore, node F broadcasts the data packet (F $\longrightarrow$ J), and all critical nodes rebroadcast (shown in Figure 7.2(a)) because none of them has a route to node J. The CDS property of critical nodes ensures delivery to node J. Critical nodes follow up the broadcast with RR to SDNC (shown in Figure 7.2(b)). Each critical node forwards the RR message just once. The SDNC selects routes to node J, includes them in an RU message, and does a network-wide broadcast (shown in Figure 7.2(c)). All nodes receive the message and learn their route to node J. Figure 7.2(d) shows all nodes being able to unicast subsequent data packets to node J.



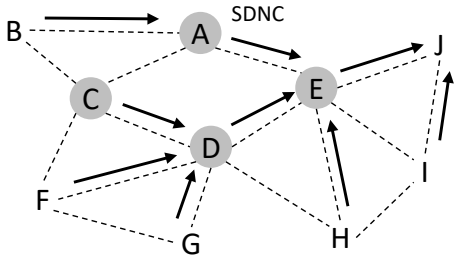
(a) Node F broadcasts a packet to node J and all critical nodes rebroadcast



(b) Critical nodes send RRs to SDNC via their RTS



(c) SDNC broadcasts RU via critical nodes (network-wide broadcast)



(d) All nodes can unicast subsequent data packets to node J

Figure 7.2: All nodes learn routes to node J. The circles around nodes represent broadcast transmissions, while the arrows represent unicast.

7.2 Communication Complexity Analysis

We now describe CORR's communication complexity (CC). We define CC as the total bytes transmitted in the network over its entire operation. Table 7.1 lists all symbols used for expressing the CC. For this analysis, we assume that each transmission has a successful delivery.

The SDNC periodically learns the network topology, which involves flooding a TD message. If R_{td} is the topology discovery rate, and N_c is the number of critical nodes, then Equation 7.1 represents CORR's periodic CC.

$$CC_{prdc} \leq R_{td}(NH_{td} + N_c(H_{ni} + M)D) \tag{7.1}$$

Here, the first term represents the CC of flooding a constant-size TD (i.e., H_{td}). The second term represents the CC of sending the neighbor information. Here, N_c

Table 7.1: Symbols

Category	Symbol	Meaning
Network	**N**	Network Size
	D	Network Diameter
	M	Average Node Degree
	B	Data Packet Size
	N_c	Critical Nodes
	R_{gen}	Data Packet Generation Rate
Messages	H_{td}	Topology Discovery Header Size
	H_{ni}	Neighbor Information Header Size
	H_{rr}	Route Requests Header Size
	H_{ru}	Route Update Header Size
Protocol	R_{td}	Topology Discovery Rate

nodes send their neighbor list having M nodes in a constant-size header (H_{ni}) and up to D nodes forward the message.

Nodes unicast or broadcast a data packet based on their route availability. If a valid route is not available and the node broadcasts, then the data packet may be forwarded by all critical nodes in the worst case. Non-critical nodes do not participate in forwarding (relaying) data packets. When a critical node broadcasts, it also sends an RR message, which may get forwarded by all critical nodes in the worst case. On receiving an RR message, the SDNC selects routes for all critical nodes and performs a network-wide broadcast. Thus, Equation 7.2 represents the CC of a route request procedure.

$$CC_{rr} \leq BN_c + N_c H_{rr} + (H_{ru} + N_c)N_c \tag{7.2}$$

Here, the first term represents the CC of flooding the data packet of size B via N_c nodes. The second term represents the CC of sending RR of size H_{rr} and N_c nodes forwarding the message in the worst case. The third term shows the CC of sending an

RU (containing N_c routes and a constant-size header H_{ru}) and N_c nodes forwarding the message.

Let p_n be the probability that routes have not been selected to the destination of the n^{th} data packet originated in the network. If they have been selected previously, then let p_{lb} be the probability of a link break. Then using Equation 7.3, we get the CC of sending the n^{th} data packet.

$$CC_n \leq (p_n + (1 - p_n)p_{lb})CC_{rr} + (1 - p_n)(1 - p_{lb})BD \tag{7.3}$$

The n^{th} data packet initiates the route request procedure with the probability $(p_n + (1 - p_n)p_{lb})$. This procedure involves flooding the data packet via the critical nodes and delivering it to its destination. Nodes use the available routes and unicast the data packet with the probability $(1 - p_n)(1 - p_{lb})$.

Assuming that the data packets are sent to destinations selected uniformly at random, then the probability of not sending a packet to a particular destination (i.e., not selecting a particular node in the network) is $(N - 2)/(N - 1)$. Let p' represent that probability.

If a data packet to the destination d initiates the route request procedure, then the SDNC selects routes to d, and sends to all nodes. Thus, the subsequent data packet will initiate the route request procedure if and only if its destination is neither d nor a neighbor node of the source. If p_{nbr} is the probability of a node being a neighbor, R_{gen} is the data packet generation rate, and T is the interval used for learning the network topology, then Equation 7.4 shows the probability that routes to the destination of the n^{th} packet would not have been previously selected.

$$p_n = ((1 - p_{nbr})p')^{(n-1) mod R_{gen}T} \tag{7.4}$$

Routes to any previous destination would have been selected by the SDNC if the destination was neither a neighbor of the source nor a previously selected destination. Here, $mod\, R_{gen}T$ is used because a new set of critical nodes are selected every T

seconds and the previously learned routes are dropped. Further, as n $\rightarrow R_{gen}T$, then $p_n \rightarrow 0$, showing that as traffic flows, the probability of routes not being available is infinitesimal. Using Equations 7.1 and 7.3 we get CORR's total CC.

$$CC_{corr} \leq CC_{prdc} + \sum_{n=1}^{\infty} CC_n \tag{7.5}$$

We compare the average CC obtained from Equation 7.5 to the average CC of an ns3 simulation experiment. In this experiment, each node sends a data packet to a destination selected uniformly at random in a 6x6 Manhattan grid. This experiment uses wireless links configured in ad hoc mode and susceptible to interference, packet losses, and collisions. So, unlike the theoretical analysis, the transmissions may fail. The average CC from Equation 7.5 is calculated using different link break probabilities (p_{lb} = 0, 0.1, 0.2, 0.3) in Equation 7.3. The other parameters (e.g., network size, payload) remain the same as in the ns3 simulation experiment.

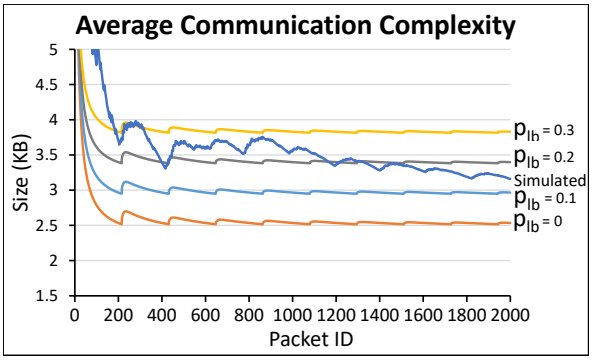


Figure 7.3: Comparison of average communication complexities from an ns3 simulation experiment and Equation 7.5 using different link break probabilities.

Figure 7.3 shows a comparison of the average CC for the first 2000 data packets transmitted in the network, where "Simulated" represents the ns3 results, and the link break probability values represent the numerical results from Equation 7.5. The average CC is high for the first few packets because nodes broadcast most of the data packets, but it quickly converges because the SDNC configures the routes for unicast transmissions. The ns3 experiment experiences several failed transmissions, resulting

Table 7.2: Simulation Scenarios

Simulation Scenario	**Network Size** (nodes)	**Network Density** (nodes/km^2)	**Data Packet Interval** (seconds)	**Node Speed** (m/s)
Increasing Network Size	[60, 100]	3.3	10	4
Decreasing Network Density	100	[11, 2]	10	4
Increasing Network Load	100	3.3	[10, 2]	4
Increasing Node Speed	100	3.3	10	[2, 10]

in either failed route updates from the SDNC or invalidating existing ones. Thus, a few nodes may continue to broadcast data packets, resulting in several critical nodes forwarding them. Here, the simulated average CC is between the numerical results obtained using 0.1 and 0.2 link break probabilities. The similarities between simulated and numerical values indicate the analysis presented in this section correctly describes CORR's CC.

7.3 Simulation Results

We now describe the simulation results of CORR for several scenarios. Since CORR is a reactive protocol, we compare the results to those of AODV [90]. Tables 7.2 and 7.3 list the simulation scenarios and parameters, respectively.

AODV simulations are performed using two different Hello intervals, namely 1 and 60 seconds. The recommended Hello interval in the AODV RFC [90] and ns3 is 1 second. However, we have found that the performance of AODV improves in our settings if we use large Hello intervals (and other associated parameters) because the network load reduces. The set of parameters providing improved AODV performance is shown under the "AODV-modified" column of Table 1. Hereafter, the RFC default and improved versions of AODV will be referred to as AODV-default and AODV-modified, respectively.

The simulation results are compared using the following three metrics: (1) Packet Delivery Ratio (PDR), which is the ratio of the total data packets received and

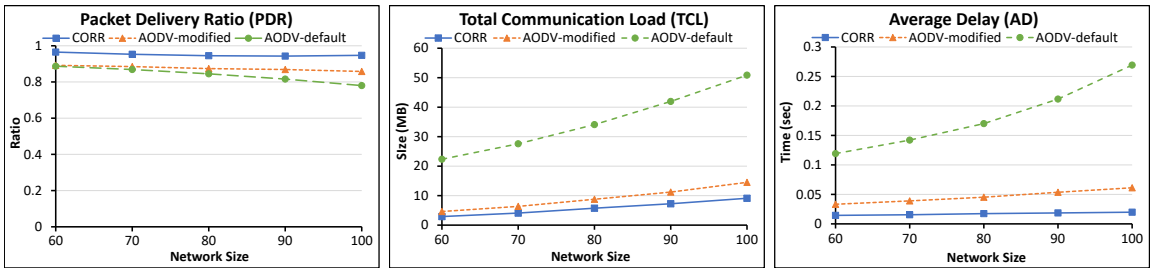
Table 7.3: Simulation Parameters

Parameter	**Value**	**Parameter**	**Value**
Simulation Time	60 minutes	Node Speed	4 m/s
Data Packet Size	200 Bytes	Node Mobility	Rand. Waypoint
Application Nodes	Rand. Src. Dst.	Trans. Power	12 dBm
Propagation Loss	Friis Model	MAC	802.11b
CORR			
TD Interval	60 secs	NbrMaintenance	Data pkt interval
AODV-default		**AODV-modified**	
Hello Interval	1 sec	Hello Interval	60 secs
Active Route	3 secs	Active Route	180 secs
MyRoute Timeout	6 secs	MyRoute Timeout	360 secs

transmitted, (2) Total Communication Load (TCL), which is the sum of the routing overhead caused by the control packets and size of the data packets, and (3) Average Delay (AD), which is an average end-to-end delay of the data packets. The result shown is an average of 10 runs.

All numerical comparisons to AODV are with the better-performing AODV-modified unless mentioned otherwise.

7.3.1 Increasing Network Size



(a) Packet Delivery Ratio (b) Total Communication Load (c) Average Delay

Figure 7.4: Simulation results for scenario 1 (Increasing Network Size) where network size ranges from 60 to 100 nodes but density remains constant.

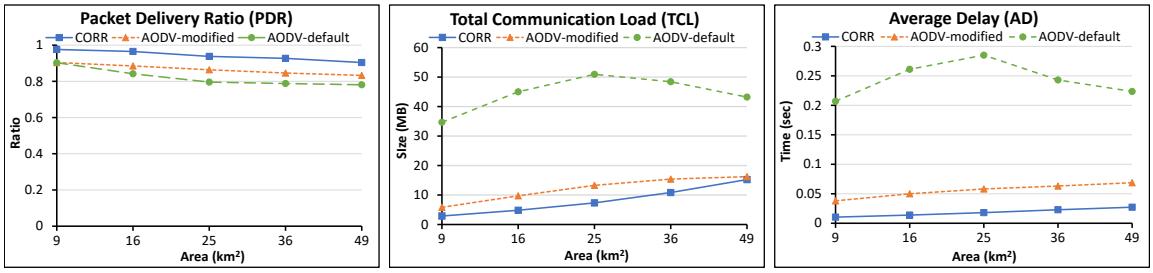
Figure 7.4 shows the results for the increasing network size scenario. CORR has up to 10% better PDR than AODV (shown in Figure 7.4(a)) for all network sizes. When valid routes are available, nodes unicast data packets to their destination. However, when topology changes and configured routes become invalid, or when routes expire, nodes start broadcasting packets. In this situation, nodes rely on critical nodes to forward, either as unicast or broadcast, but ensure successful deliveries to the destinations. Only critical nodes forward (both control and data), keeping the overall interference and collisions low – both these factors result in a high PDR. On the other hand, AODV employs extensive use of control packets for its route discovery, route maintenance, and local connectivity procedures. Each node participates in either originating or forwarding the control packet, resulting in high interference and collisions for data packets and the network experiences a low PDR.

Despite broadcasting data packets when no routes are available, CORR has ∼1.6x lower TCL than AODV (as shown in Figure 7.4(b)). It indicates that as the network topology changes, the SDNC rapidly updates routes in nodes for unicast transmissions. Further, only a single route request results in SDNC updating routes in all nodes. Note that nodes receive routing information only to the destination in the request message. Thus, the routing overhead and the TCL remains low. In AODV, the route request procedure results in all nodes learning reverse routes to the source (but only in the case of full flooding). Whereas, the intermediate nodes (between the source and destination nodes) learn forward routes to the destination. Thus, the route request procedure may get initiated several times for a particular destination, causing high routing overhead and high TC.

Figure 7.4(c) shows CORR having up to ∼3x lower AD than AODV because data packets are not buffered. Instead, nodes broadcast them and expect critical nodes to deliver them to their destination. In AODV, a node buffers the data packet until it learns a route. If a node fails in the first attempt, it reinitiates the route discovery procedure, keeping the packet buffered. High routing overhead contributes to high collisions and several MAC layer back-offs and retransmissions. Both of these factors

result in a high average delay.

7.3.2 Decreasing Network Density



(a) Packet Delivery Ratio (b) Total Communication Load (c) Average Delay

Figure 7.5: Simulation results for scenario 2 (Decreasing Network Density) where network size is 100 nodes but density ranges from approx. 11 to 2 nodes/km^2 (simulation area between 9 km^2 and 49 km^2).

Figure 7.5 shows results for the decreasing network density scenario. CORR has at least 8% better PDR than AODV in both dense (9 km^2) and sparse (49 km^2) networks (shown in Figure 7.5(a)). In dense networks, both AODV and CORR benefit from nodes being in close proximity because the Hello exchanges (in AODV) and Topology Discovery (in CORR) result in each node learning routes to a majority of other nodes (i.e., neighbors). CORR benefits more because only a few critical nodes are selected, resulting in a low load – thus, low interference and collisions – and a high PDR. However, AODV experiences packet losses, and hence, a low PDR, because a route discovery procedure results in route replies from a majority of the nodes, increasing both interference and collisions.

Sparse networks are more susceptible to link breaks, affecting both data and control packets, and hence, invalidating several routes. CORR switches to broadcasting data packets, which are forwarded by all the critical nodes. The redundant transmissions by critical nodes help maintain a high PDR. AODV depends solely on unicast transmissions to the next hops. So, frequent link breaks result in several packet drops. Although the local connectivity procedure detects link breaks, it takes two failed Hello

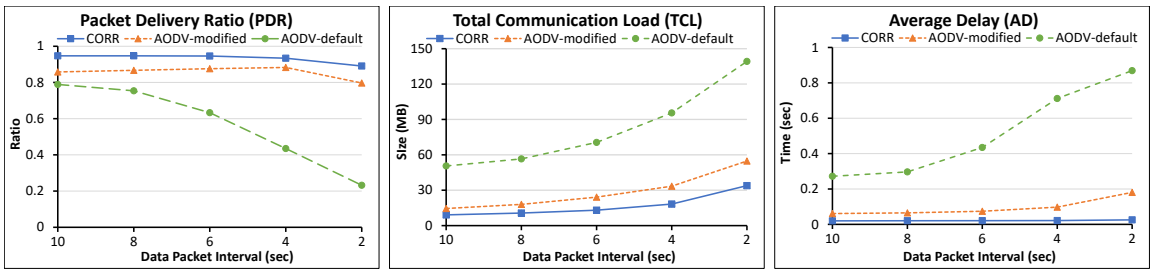
messages to confirm one [90]. The control packet transmissions are also affected by the link breaks, resulting in failed attempts to learn routes.

Figure 7.5(b) shows the TCL results. In dense networks, both AODV and CORR have low TCLs because AODV initiates the route discovery only for a few distant nodes, and CORR selects only a few critical nodes. Further, both data and control packets travel fewer hops because nodes are in close proximity. As the density decreases, packets travel more hops in general, so the TCL increases. However, AODV's TCL stabilizes because the number of nodes replying to the route requests decreases, so the control packet exchange reduces. On the other hand, CORR experiences an increase because more critical nodes are selected, and the average distance between the critical nodes and the SDNC increases. Hence, RUs and NIs travel more hops, resulting in higher routing overhead. For the sparsest network, both CORR and AODV have the same TCL, but until the sparsest network threshold, CORR maintains up to 1.5x lower TCL than AODV.

Figure 7.5(c) shows both CORR and AODV having low ADs in dense networks because the data packets travel only a few hops, but as the density decreases, the delay increases. However, CORR maintains up to 3x lower delay than AODV for the same reasons described earlier in Section 7.3.1.

7.3.3 Increasing Network load

Figure 7.6 shows the results for the increasing network load scenario. Increasing the traffic load (i.e., decreasing the data packet interval) is likely to increase interference and collisions in the network. Figure 7.6(a) shows CORR experiencing a drop in the PDR as the network load increases. AODV's PDR increases until a threshold (4 seconds) but drops after that. AODV always unicasts the data packets. So, if routes are valid (i.e., no link breaks), sending data packets at a higher rate improves the PDR. When the traffic load goes high enough, the resulting interference starts causing packet losses. Further, the network that was already congested and experienced a packet loss becomes overwhelmed with control packets transmitted during route discovery, route



(a) Packet Delivery Ratio (b) Total Communication Load (c) Average Delay

Figure 7.6: Simulation results for scenario 3 (Increasing Network Load) where network size is 100 but data packet interval ranges from 10 to 2 seconds.

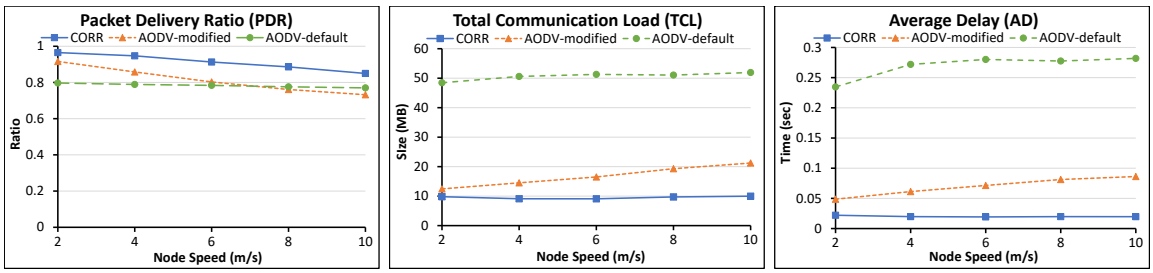
maintenance, and local connectivity procedures – further reducing the PDR. Despite having different behaviors, CORR maintains at least 5-10% better PDR than AODV for all traffic loads.

Figure 7.6(b) shows that as the network traffic increases, the TCL also increases, mainly because of the nodes forwarding more data packets. However, irrespective of the traffic rate, CORR always maintains up to 1.6x lower TCL than AODV. The increasing routing overhead in AODV keeps the TCL high because every node that identifies a link break either initiates link failure reporting or route discovery procedure. In contrast to AODV, only critical nodes send route requests and forward the control packets in CORR, keeping the TCL low.

Figure 7.6(c) shows both CORR and AODV having low delays at low traffic load, but as the traffic grows, AODV's delay increases significantly. For the highest traffic rate (i.e., 2 seconds interval), CORR has ∼7x lower delay than AODV mainly due to no buffering and fewer MAC-layer backoffs and retransmissions.

7.3.4 Increasing Node Speed

Figure 7.7 shows the results for the increasing node speed scenario. Similar to the network load, increasing the node speed is likely to increase packet losses (both control and data) but due to frequent link breaks. Figure 7.7(a) shows the decreasing PDR for both CORR and AODV. However, CORR continues to maintain up to 10%



(a) Packet Delivery Ratio (b) Total Communication Load (c) Average Delay

Figure 7.7: Simulation results for scenario 4 (Increasing Node Speed) where network size is 100 but node speed ranges from 2 to 10 m/s.

better PDR than AODV because the nodes switch to broadcast transmissions in the absence of valid routes. As a result, several critical nodes forwarding data packets and ensuring their delivery. On the other hand, link breaks in AODV cause several nodes to send link failure notifications and initiate route discovery procedures, increasing the routing overhead significantly (shown in Figure 7.7(b)). Despite broadcasting the data packets, CORR's TCL remains up to ∼2x lower than AODV's TCL because the network size and the number of critical nodes remain the same. Interestingly, AODV-default has a better PDR than AODV-modified in high mobility scenarios because of its better link connectivity maintenance. However, AODV-default's TCL is significantly higher than that of AODV-modified.

Figure 7.7(c) shows that both CORR and AODV have low delays in low mobility scenario, but as mobility increases, AODV's delay increases significantly. CORR has up to 4x lower delay than AODV in the high mobility scenarios. Delay increases in AODV because nodes buffer the data packets until new routes are available after link breaks.

7.4 Conclusions

In this chapter, we have presented a centralized reactive routing protocol called CORR for our SD-MANET architecture. It addresses the limitations of the first routing protocol described in Chapter 4 by employing the features of the ECHO and VINE

protocols.

Although designed for network-wide broadcasts, CORR leverages the ECHO protocol for learning the network topology and creating a strong network backbone of nodes for unicast forwarding of the data packets. Only the critical nodes forward requests and routing information to and from the SDNC, becoming the network backbone and reducing the control overhead, interference, and collisions, and improving the scalability. Further, the CDS property of critical nodes ensures that a network-wide broadcast from the SDNC containing the routing information reaches all nodes.

CORR also leverages the IA feature of the VINE protocol for transmitting control messages and increases the per-hop reliability of the NI messages and likelihood of SDNC learning a connected network topology.

We have evaluated CORR extensively using ns3 simulations and compared the results to those of AODV for a variety of network scenarios, addressing scalability, load, density, and mobility. CORR outperforms AODV in all scenarios, providing better reliability and resiliency against dynamic topology changes, proving to be a better protocol, in general, for all network scenarios. It provides up to 10% better delivery ratio than AODV while incurring at least 1.5x lower network load and 3x lower delay. A low network load helps improve the lifespan of the mobile nodes, while a low delay makes CORR suitable for time-sensitive applications.

Chapter 8

CENTRALIZED PROACTIVE ROUTING

In this chapter, we present a protocol, called Centralized Proactive Routing (CPR), designed for the SD-MANET architecture described in Chapter 3. Similar to the CORR protocol described in Chapter 7, CPR uses critical nodes for learning the network topology and disseminating the routing information.

Chapter 4 described the PCC protocol and the results of its comparison with OLSR and DSDV. At the end of that chapter, we discussed a few limitations that prevent PCC from attaining better results than OLSR in large networks.

Chapter 7 described the CORR protocol that addresses PCC's limitations. The use of critical nodes for learning the network topology and efficiently disseminating the routing information reduces the communication overhead significantly. Further, the use of implicit acknowledgments for sending neighbor information allows SDNC to learn a connected network topology.

CORR is a reactive protocol, so we compared its results to those of AODV, which is the state-of-the-art *reactive* routing protocol. This chapter presents CPR, which is a proactive protocol, so we compare its results to those of OLSR, which is the state-of-the-art *proactive* routing protocol.

8.1 The CPR Protocol

Unlike PCC and CORR, we describe only the Sending Network Routes function of CPR because its other two functions are the same as those of CORR. In CORR, the SDNC sends network routes *reactively*, whereas in CPR, the SDNC sends network routes *proactively*.

8.1.1 Sending Network Routes

The SDNC selects critical nodes and learns the network topology periodically after every $TDInterval$ to account for network dynamics. Then it selects routes for *all* critical nodes such that they can send data packets to every other node in the network. The SDNC disseminates the routing information as network-wide broadcasts.

Algorithm 10 Sending Network Routes And Forwarding Data Packets

```
1:  procedure SendingNetworkRoutes
2:      for each critical node c do
3:          for each destination d do
4:              r ← route between c and d
5:              Include r in RU
6:          end for
7:      end for
8:      RU.seqNum ← RUSeqNum++
9:      Broadcast RU
10: end procedure
11: procedure ForwardDataPacket(pkt)
12:     if state == critical or node == pkt.source then
13:         if route to pkt.dst is valid then
14:             Unicast pkt to the nextHop
15:         else if state is non-critical and RTS is valid then
16:             Unicast pkt to RTS
17:         else
18:             Broadcast pkt
19:         end if
20:     end if
21: end procedure
```

Algorithm 10 describes the procedures used for forwarding the data packets and sending the network routes. The SDNC calls the *SendingNetworkRoutes* procedure periodically for selecting the routes. However, nodes may invalidate their configured routes on detecting link breaks. A link break is detected when the node fails to receive a packet from the neighbor in the past *NbrMaintenance* interval. Nodes may also detect link breaks using the 802.11 CSMA/CA schemes (i.e., RTS/CTS/ACK), if available. If a valid route is not available for forwarding the data packet, then the packet is broadcast. Both these features are similar to CORR's.

The *ForwardDataPacket* procedure describes the rules for forwarding data packets. However, unlike in CORR, the critical nodes in CPR do not send route request messages to the SDNC. Instead, they continue to forward either as unicast or broadcast, depending on the route validity, relying on the CDS property for delivering the packets to their destination.

8.2 Communication Complexity

We now describe CPR's communication complexity (CC). Similar to the analyses presented in previous chapters, we define CC as the total bytes transmitted in the network during its entire operation. Table 8.1 lists all symbols used for expressing the CC. For this analysis, we assume that each transmission has a successful delivery.

Table 8.1: Symbols

Category	Symbol	Meaning
Network	**N**	Network Size
	D	Network Diameter
	M	Average Node Degree
	B	Data Packet Size
	R_{gen}	Data Packet Generation Rate
	N_c	Critical Nodes
Messages	H_{td}	Topology Discovery Header Size
	H_{ni}	Neighbor Information Header Size
	H_{ru}	Route Update Header Size
Protocol	R_{td}	Topology Discovery Rate

CPR uses three control messages and the complexity of each of them is described in Table 8.2. Equation 8.1 represents the combined control communication complexity of all control messages, where R_{td} is the topology discovery rate.

$$CC_{ctrl\_cpr} \leq R_{td}(NH_{td} + (H_{ni} + M)N_cD + (H_{ru} + N)N_c^2) \tag{8.1}$$

Table 8.2: Communication complexity of each message

Message	Complexity	Explanation
TD	NH_{td}	SDNC broadcasts a TD message of size H_{td}, and each node rebroadcasts it.
NI	$(H_{ni} + M)N_cD$	Each critical node sends the information of its M neighbors in an NI message to the SDNC. The network diameter is D, so up to D nodes may forward.
RU	$(H_{ru} + N)N_c^2$	SDNC sends RU messages, one for each of its N_c critical node. Each such message has routes to every node in the network (i.e., N routes). Each such message is a network-wide broadcast and gets forwarded by N_c nodes.

We ignore the constants: H_{td}, H_{ni}, H_{ru}, and R_{td} for analyzing CPR's asymptotic control communication complexity (ACCC). If the average node degree is d, i.e., $M = d$, the ACCC of CPR is $O(N+dN_cD+NN_c^2)$, which essentially is $O(dN_cD+NN_c^2)$. Thus,

$$CC_{ctrl\_cpr} = O(dN_cD + NN_c^2) \tag{8.2}$$

Similar to PCC, CPR is a proactive protocol, so the SDNC periodically updates the routing tables of all nodes with routes to every node in the network. However, unlike PCC, if a node invalidates routes (on detecting a link break), it continues to forward but as broadcast instead of unicast. If p_{lb} is the probability of a link break between source and destination, then Equation 8.3 shows the CC of a data packet. If the route is valid, then up to D nodes may forward, else we consider it to a network-wide broadcast, in which up to N_c nodes forward.

$$CC_{data\_cpr} \leq (1 - p_{lb})BD + p_{lb}BN_c \tag{8.3}$$

Considering R_{gen} as the data packet generation rate, the total communication

complexity of CPR is:

$$CC_{cpr} \leq CC_{ctrl\_cpr} + R_{gen} CC_{data\_cpr} \tag{8.4}$$

We now analyze CPR's ACCC for dense and sparse networks. For dense networks, the average node degree d is very high, and the network diameter D is very small, so $d = O(N)$ and $D = O(1)$. Further, a node degree of $O(N)$ results in the selection of only a few critical nodes, hence $N_c = O(1)$. Substituting these values in the ACCC shown in Equation 8.2, we get $O(N + N)$, which is $O(N)$.

For sparse networks, the node degree d is very small, so we consider $d = O(1)$, which makes the critical nodes and the network diameter grow in the order of N. Thus we consider $N_c = O(N)$ and $D = O(N)$. If we substitute these values in Equation 8.2, we get $O(N^2 + N^3)$, which is $O(N^3)$.

Table 8.3: Comparison of the asymptotic control communication complexities

	PCC	**CPR**
Generic	$O(dND + N^2D)$	$O(dN_cD + NN_c^2)$
Dense	$O(N^2)$	$O(N)$
Sparse	$O(N^3)$	$O(N^3)$

Table 8.3 shows a comparison of the ACCC of PCC and CPR for generic, dense, and sparse networks. For the generic networks with node degree d, the relative gain of CPR over PCC increases with decreasing N_c. The complexity remains the same in sparse networks, but CPR attains O(N) better (lower) complexity than PCC in dense networks.

8.3 Simulation Results

We now describe the simulation results of CPR. We compare CPR's results to those of OLSR. Table 8.4 lists the simulation scenarios, which are similar to those used

Table 8.4: Simulation Scenarios

Network Scenario	**Network Size** (nodes)	**Network Density** (nodes/km^2)	**Data Packet Interval** (seconds)	**Node Speed** (m/s)
Increasing Network Size	[60, 100]	3.3	10	4
Decreasing Network Density	100	[11, 2]	10	4
Increasing Network Load	100	3.3	[10, 2]	4
Increasing Node Speed	100	3.3	10	[2, 10]

for evaluating CORR. Table 8.5 lists all the simulation parameters.

Table 8.5: Simulation Parameters

Parameter	**Value**	**Parameter**	**Value**
Simulation Time	300 seconds	Node Speed	4 m/s
Data Packet Size	200 Bytes	Node Mobility	Rand. Waypoint
Application Nodes	Rand. Src. Dst.	Trans. Power	12 dBm
Propagation Loss	Friis Model	MAC	802.11b
CPR			
TD Interval	30 secs	NbrMaintenance	Data pkt interval
OLSR			
Hello Interval	[2, 6] seconds	TC Interval	5 seconds

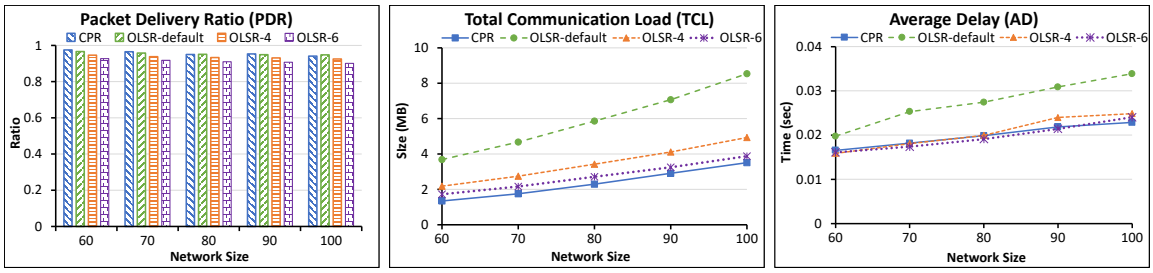
The simulation results are compared using the following three metrics: (1) Packet Delivery Ratio (PDR), which is the ratio of the total data packets received and transmitted, (2) Total Communication Load (TCL), which is the sum of the routing overhead caused by the control packets and size of the data packets, and (3) Average Delay (AD), which is an average end-to-end delay of the data packets. The result shown is an average of 10 runs.

The recommended Hello interval in the OLSR RFC [35] and ns3 is 2 seconds, but it creates a high communication overhead. So, we have included OLSR results with several different Hello intervals, namely 2, 4, and 6 seconds, and the plots show their

results as OLSR-2, OLSR-4, and OLSR-6, respectively. A common trend observed in all simulation results of all scenarios is that a larger Hello interval results in reducing the control overhead but also reduces the PDR. The higher the interval, the lower the control overhead and the PDR. Intuitively, a high Hello interval results in a less frequent update of network routes, so nodes experience many packet losses and the PDR drops. On the other hand, a low Hello interval results in increasing the average delay for data packets because the CSMA causes repeated back-offs and retransmissions.

We have shown the results of all the Hello intervals in plots, but for brevity, numerical comparisons in the description are included only for OLSR-2.

8.3.1 Increasing Network Size



(a) Packet Delivery Ratio (b) Total Communication Load (c) Average Delay

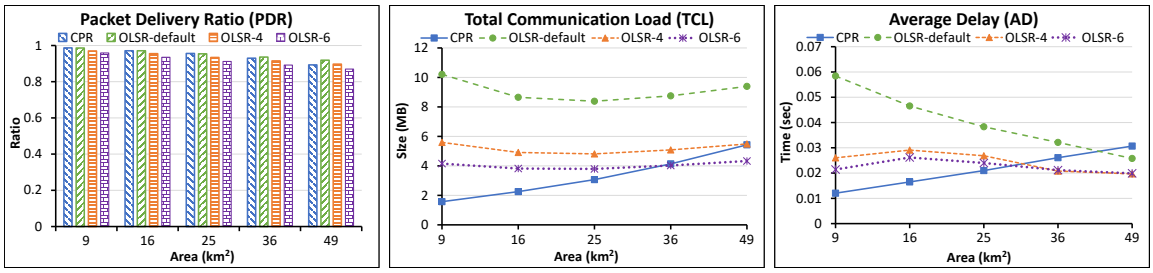
Figure 8.1: Simulation results for scenario 1 (Increasing Network Size) where network size ranges from 60 to 100 nodes but density remains constant.

Figure 8.1 shows the results for the increasing network size scenario. CPR has a better or the same PDR compared to OLSR-2 for up to network size 100 (shown in Figure 8.1(a)). The SDNC periodically updates network routes in all nodes, allowing them to unicast data packets to their destination. If the nodes invalidate their routes due to link breaks, CPR still attains a high PDR by allowing nodes to broadcast packets. The CDS property of critical nodes ensures delivery of data packets to their destination.

Using only critical nodes for learning network topology and disseminating network routes, CPR keeps the TCL low. On the other hand, OLSR-2's frequent and

large-sized Hello message transmissions result in a high TCL. Figure 8.1(b) shows CPR having up to 2.4x lower (better) TCL than OLSR-2. The large-sized Hello messages also result in repeated MAC-layer back-offs and retransmissions for data packets. Figure 8.1(c) shows CPR having up to 1.4x lower AD than OLSR-2.

8.3.2 Decreasing Network Density



(a) Packet Delivery Ratio (b) Total Communication Load (c) Average Delay

Figure 8.2: Simulation results for scenario 2 (Decreasing Network Density) where network size is 100 nodes but density ranges from approximately 11 to 2 nodes/km^2 (i.e., simulation area from 9 to 49 km^2)

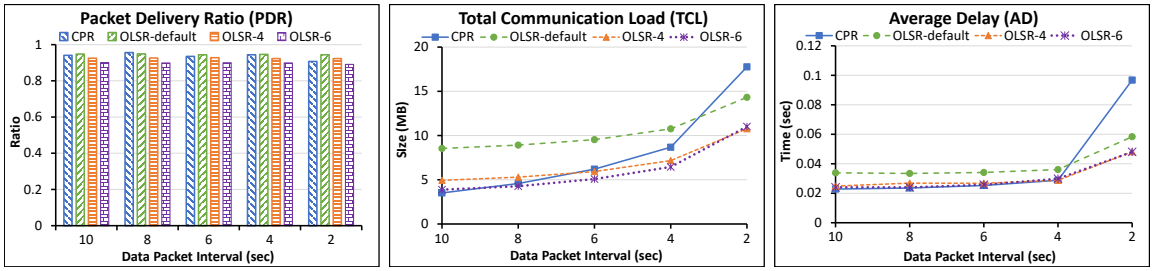
Figure 8.2 shows the results for the decreasing network density scenario. In dense networks, both OLSR and CPR benefit from nodes being in close proximity because only a few critical nodes (in CPR) and MPR nodes (in OLSR) are selected. As a result, both protocols attain high PDRs (shown in Figure 8.2(a)).

Sparse networks experience frequent link breaks, which affects both data and control packets, invalidating several routes. As the density decreases and the average distance between the nodes increases, the PDR starts to drop. In the sparsest network, CPR's PDR is lower than that of OLSR-2 because frequent link breaks result in several nodes invalidating their routes and broadcasting data packets. As a result, a few of them fail to reach their destination, resulting in a low PDR. Figure 8.2(b) confirms that by showing a steeper increase in CPR's TCL over OLSR-2's. Despite the steeper increase, CPR's TCL remains up to 1.7x lower than that of OLSR-2 even for the sparsest network. In dense networks, both CPR and OLSR-2 have low TCLs, but as

the density decreases the TCL increases for both. An increase in the average hops between the nodes also increases the TCL.

Figure 8.2(c) shows OLSR-2 and CPR having contrasting results. In dense networks, the large-sized Hello packets result in frequent MAC-layer back-offs and retransmissions for data packets, so OLSR-2 has higher AD than CPR. As the density decreases, a smaller node degree (i.e., number of neighbors) reduces the size of Hello packets and also the AD. On the other hand, CPR's AD increases with the decreasing density because nodes broadcast more data packets due to frequent link breaks.

8.3.3 Increasing Network Load



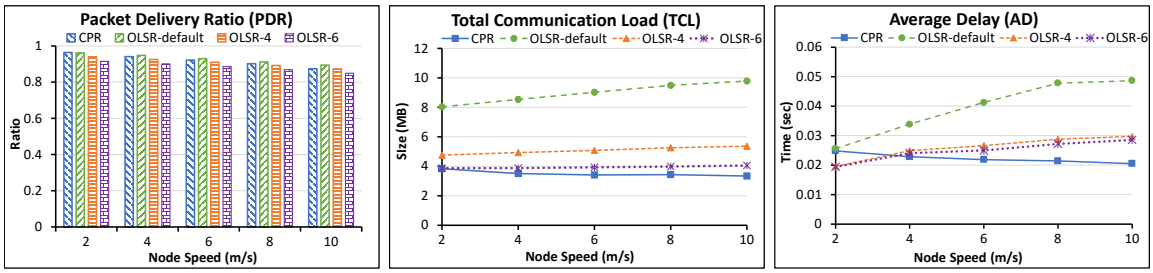
(a) Packet Delivery Ratio (b) Total Communication Load (c) Average Delay

Figure 8.3: Simulation results for scenario 3 (Increasing Network Load) where network size is 100 but data packet interval ranges from 10 to 2 seconds.

Figure 8.3 shows the results for the increasing network load scenario. Figure 8.3(a) shows CPR and OLSR-2 having almost the same PDR for all intervals, except the smallest. At high network loads (i.e., at smaller intervals), nodes transmit more data packets as broadcast in the absence of valid routes, resulting in a high TCL. Figure 8.3(b) confirms that by showing a steeper increase in CPR's TCL over OLSR-2's, yet CPR's TCL is ∼1.2x lower than that of OLSR-2 for the highest load (i.e., the interval of 4 seconds). Figure 8.3(c) shows increasing AD with increasing network load for both CPR and OLSR because frequent data packets transmissions cause more MAC layer back-offs and retransmissions. Thus, at very high loads, CPR's PDR drops

and AD increases because of several broadcast transmissions in the network. However, at lower loads, CPR provides the same PDR as OLSR-2 with lower TCL and AD.

8.3.4 Increasing Node Speed



(a) Packet Delivery Ratio (b) Total Communication Load (c) Average Delay

Figure 8.4: Simulation results for scenario 4 (Increasing Node Speed) where network size is 100 but node speed ranges from 2 to 10 m/s.

Figure 8.4 shows the results for the increasing node speed scenario. Increasing node speed is likely to increase packet losses (both control and data) due to frequent link breaks. Figure 8.4(a) shows PDR decreasing for all with the increasing node speeds. CPR's PDR remains almost the same as that of OLSR for low node speed but drops at higher node speeds.

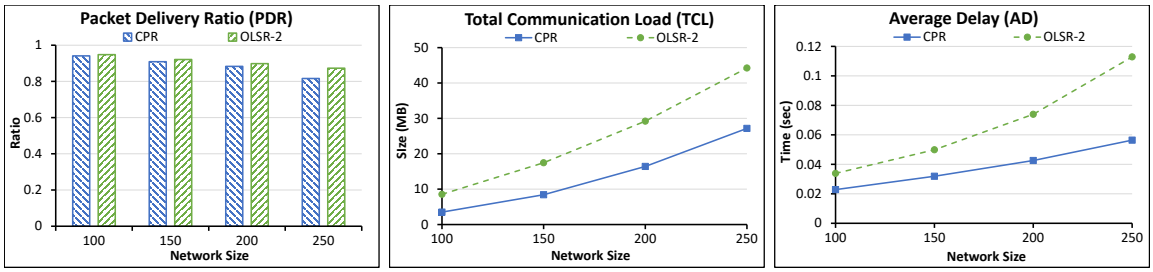
The node speed has a contrasting effect on the TCLs of CPR and OLSR-2. The Random Waypoint mobility model at high node speeds concentrates more nodes at the center of the simulation region. As a result, nodes are in closer proximity to each other because of which CPR selects fewer critical nodes, and OLSR transmits larger sized Hello packets. Figure 8.4(b) confirms that by showing a decrease in CPR's TCL but an increase in OLSR-2's with increasing node speeds. The selection of fewer critical nodes results in reducing communication overhead of learning network topology and sending network routes. As a result, the CPR's TCL reduces. Irrespective of the node speed, the CPR's TCL remains at least 2x lower than that of OLSR-2. CPR's decreasing delay with increasing node speed shown in Figure 8.4(c) also confirms the selection of

fewer critical nodes. CPR-2's AD is almost the same as that of OLSR-2 for node speed 2 m/s, but ∼2.3x lower for node speed 10 m/s.

8.4 Scalability Issues

The simulation scenarios considered in Section 8.3 evaluate CPR for networks of sizes up to 100 nodes. In this section, we show CPR's simulation results for networks of sizes up to 250 nodes.

Figure 8.5 shows a comparison of the results of CPR and OLSR-2 for network size between 100 and 250 nodes. As the network size increases, CPR starts experiencing a drop in the PDR (shown in Figure 8.5(a)). However, CPR's TCL and AD remain lower than those of OLSR-2 (shown in Figures 8.5(b) and 8.5(c)).



(a) Packet Delivery Ratio (b) Total Communication (c) Average Delay

Figure 8.5: Simulation results for networks of sizes up to 250 nodes.

Large networks (i.e., size ≥ 100 nodes) result in the selection of several critical nodes. Our simulation results show that CPR selects 37 critical nodes in a network of size 250 nodes. Learning the network topology from the neighbor information of a large number of critical nodes results in congestion at SDNC. Further, the SDNC ends up sending the routing information in several RU messages, resulting in many network-wide broadcasts. These results indicate that a single node (i.e., SDNC) may be inadequate for performing all the three SD-MANET functions in large networks. Therefore, solutions are needed to address the scalability issues in SD-MANET.

8.5 Conclusions

In this chapter, we presented our second centralized proactive routing protocol called CPR for the SD-MANET architecture. Similar to the CORR protocol, it addresses the limitations of the first routing protocol (PCC) described in Chapter 4 by employing the features of the ECHO and VINE protocols.

We analyzed CPR's communication complexity and proved it to be O(N) better (lower) than that of PCC in dense networks. We also showed the results of an extensive evaluation of CPR for scenarios addressing scalability, load, density, and mobility. The results show that CPR has about the same PDR as OLSR but incurs far less overhead (2.4x) and has lower latency (1.4x) for networks of size 100 nodes. However, repeating the simulation experiments over large networks (i.e., size $\geq$ 100 nodes) results in poorer performance. In particular, the PDR drops, indicating that in a centralized architecture, large networks need hierarchical solutions. We present one such solution in Chapter 9.

Chapter 9

HIERARCHICAL CENTRALIZED PROACTIVE ROUTING

In this chapter, we present a protocol, called Hierarchical Centralized Proactive Routing (HCPR), designed for the SD-MANET architecture described in Chapter 3. The HCPR protocol addresses the scalability issues of CPR discussed in Section 8.4.

The simulation results in Section 8.4 showed that CPR's performance drops for large-sized networks. In particular, CPR's delivery ratio becomes significantly lower than that of OLSR for networks of size more than 100 nodes. Since the number of critical nodes grows in the order of network size, the control messages sent by these nodes create a bottleneck at the SDNC. Further, the route updates sent by the SDNC result in several network-wide broadcasts. Thus, in a large network, a single node (i.e., SDNC) may be inadequate for performing all the necessary functions.

Here, we describe a hierarchical routing approach that continues to use a single SDNC but significantly reduces the bottleneck by forming clusters in the network. A cluster is a group of nodes having a Cluster Head (CH), which selects and disseminates routes for nodes in the cluster.

We first describe the HCPR protocol and then analyze its communication complexity. In the end, we present the simulation results of extensive evaluation of HCPR using the same set of scenarios as in the previous chapters, but for large networks, and compare them to those of OLSR.

9.1 The HCPR Protocol

The HCPR protocol is a hierarchical protocol that creates clusters in the network. Each cluster has a Cluster Head (CH). Nodes receive intra-cluster routing information from the CH. In addition to forming clusters, the HCPR protocol also selects

gateway nodes for inter-cluster routing. HCPR is designed to be proactive and performs all its operations periodically to account for network dynamics.

Similar to the previous SD-MANET routing protocols, HCPR is described using the following three functions: (1) learning route to SDNC, (2) learning network topology, and (3) sending network routes.

9.1.1 Learning Route to SDNC

The SDNC periodically broadcasts a message called Topology Discovery (TD). Similar to the TD message used by both CORR and CPR protocols, HCPR uses a TD message with fields for the sequence number and previous sender. In addition to these fields, HCPR also includes a field for the *Cluster Radius*. We use K to represent the value in this field. Nodes use this value for determining the cluster boundaries.

Nodes that receive the TD message with K $>$ 0 decrement the value of K and rebroadcast the TD message, resulting in Flooding the TD message to all nodes within a radius of K from the SDNC.

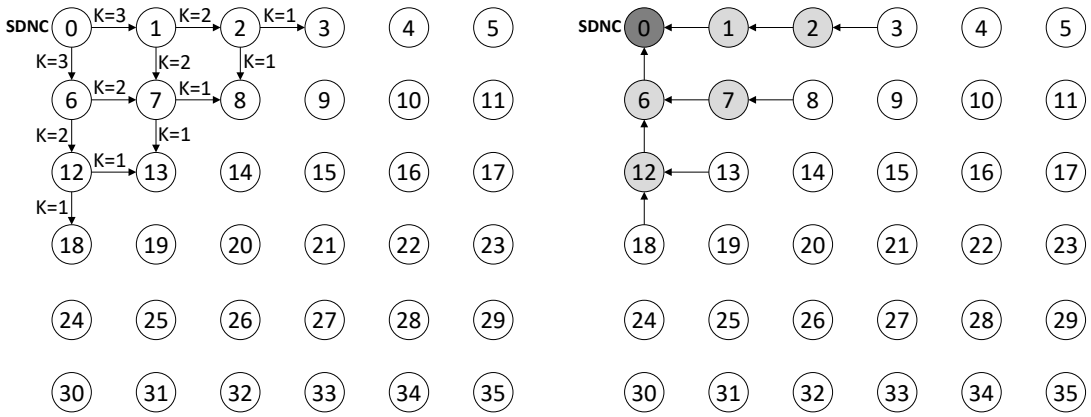
Figure 9.1(a) shows a 6x6 Manhattan Grid topology, in which the SDNC floods a TD message setting K = 3. Nodes decrement K's value before forwarding the TD message. In this example, we use K = 3 to show clusters of radius 3, but the protocol could be configured to use any value.

HCPR uses a TD flooding process which is similar to the one used by CORR and CPR. In particular, the TD flooding results in (1) nodes learning their reverse route to SDNC (RTS), (2) nodes learning their neighbors, and (3) selection of critical nodes. Figure 9.1(b) shows nodes knowing their RTS and the SDNC identifying critical nodes (shaded).

We now explain the cluster formation and selection of gateway nodes.

9.1.1.1 Cluster Formation

The initial value of K determines the cluster radius (and the cluster formation). Nodes that receive the TD message with K = 1 broadcast it by decrementing K's value



(a) SDNC flooding TD

(b) Node learning their RTS

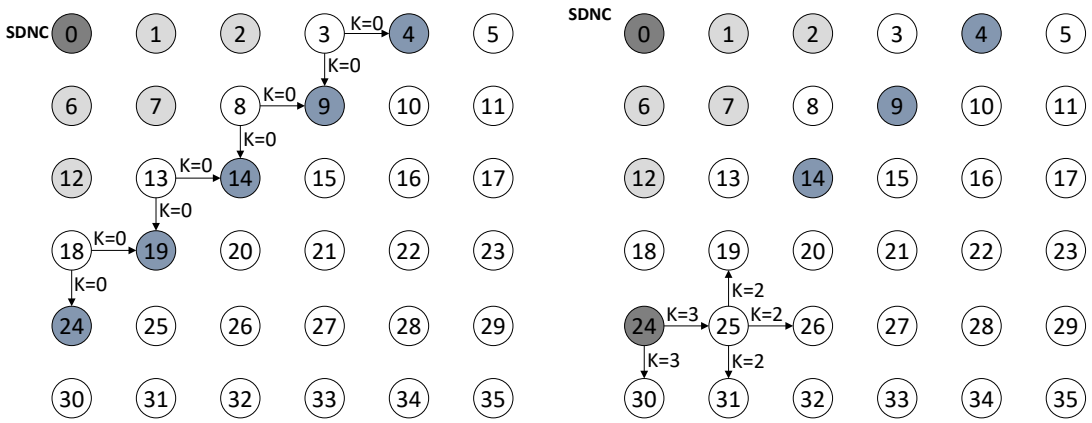
Figure 9.1: (a) SDNC initiated TD flooding with cluster diameter (K) = 3. (b) Nodes learned their Route to SDNC (RTS). Critical nodes are shown shaded.

(i.e., making K = 0). However, these nodes do not set themselves to *pending*, which is the transition state between *critical* and *non-critical*. Refer to Figure 5.2 for the state diagram of the ECHO protocol. Not setting themselves to pending essentially means that these nodes are not candidates for becoming critical.

On the other hand, the nodes that receive TD with K = 0 are candidates for becoming cluster heads of new clusters. Figure 9.2(a) shows nodes 4, 9, 14, 19, and 24 receiving TD with K = 0 and becoming CH candidates.

Each CH candidate schedules the transmission of a new TD message using a random delay[1]. The random delay is used to avoid all candidates becoming CHs and forming their cluster individually. The candidate resets K to the required cluster radius. Figure 9.2(b) shows node 24 becoming the first candidate to initiate the TD flooding procedure, thereby becoming a CH. Not all candidates become CH, only the ones that get to initiate their TD flooding procedure. If a candidate receives a TD originated at

[1] A delay chosen randomly between 0 and K*0.1 seconds.



(a) Candidate cluster heads

(b) Candidate node 24 initiating TD flooding

Figure 9.2: (a) TD message with K=0 resulting in nodes becoming candidates for Cluster Head. (b) Candidate CH node 24 randomly becomes first to initiate its TD flooding with K = 3.

another node before it transmits its own, then it will not initiate its TD flooding (or become a CH). The order in which the candidates become CHs is random and depends on the delay. In this example, the candidate node 19 receives the TD originated at node 24, and hence, becomes a part of the cluster of node 24. Node 19 is not a candidate anymore and will not initiate its scheduled TD flooding procedure.

Figure 9.3 shows candidates 14 and 4 becoming CHs and forming their respective clusters. Note that node 9 was previously a candidate CH but has now become a part of the cluster of node 14. The SDNC and node 28 form their clusters too.

Each TD flooding procedure results in the formation of a cluster and the nodes in the cluster learn their route to the CH, learn their neighbors, and select the critical nodes.

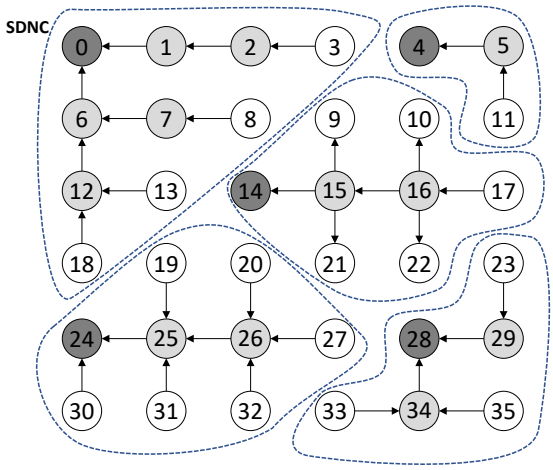


Figure 9.3: Cluster formation at the end of TD flooding.

9.1.1.2 Gateway Nodes

In addition to creating clusters, it is important to identify gateway nodes connecting clusters and enabling inter-cluster routing. Here, a gateway node is a node that connects to a CH of another cluster. When a candidate initiates the TD flooding process, it uses the same sequence number that was in the received TD message but refreshes the value of K to the cluster radius. Nodes in the neighboring clusters on receiving a duplicate TD (having the same sequence number) but K = cluster radius mark themselves to be gateway nodes. Figure 9.4 shows the gateway nodes selected in the network.

9.1.2 Learning Network Topology

SDNC and CHs need to learn the topology of their respective clusters. The algorithm used for learning is the same as Algorithm 7 used by the CORR protocol. Each critical node sends its neighbor information in a message called Neighbor Information (NI). Nodes use the Implicit Acknowledgement (IA) feature for sending the NI messages - thereby, increasing the per-hop reliability (see Section 6.2). The critical

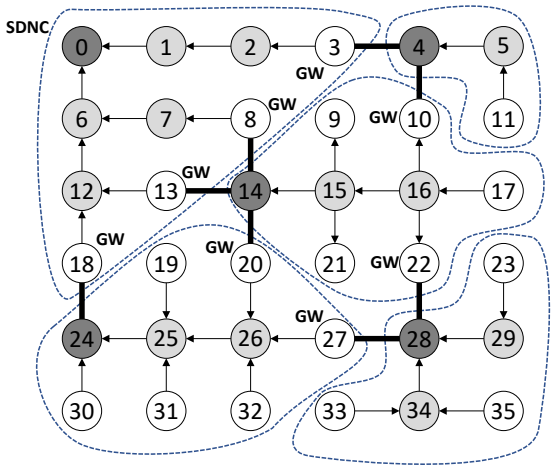


Figure 9.4: Nodes 3, 8, 10, 13, 18, 20, 22, and 27 becoming gateway nodes at the end of TD flooding.

nodes in the cluster of SDNC send their NI to the SDNC, whereas all other critical nodes send their NI to their respective CH. The CHs store the received NI messages for learning the cluster topology and do not forward these messages.

9.1.3 Sending Network Routes

HCPR uses the Route Update (RU) and Cluster Information (CI) messages for configuring intra-cluster and inter-cluster routes, respectively.

9.1.3.1 Intra-Cluster Routing

Upon learning the cluster topology from the NI messages, each CH (and SDNC) selects routes for intra-cluster routing and sends them in RU messages as network-wide broadcasts. The procedure is the same as used by the CPR protocol and described in Algorithm 10. The difference is that the routes are selected only for critical nodes in the same cluster. Figure 9.5 shows a hierarchical view of the network shown in Figure 9.1. The figure also shows CHs (and SDNC) sending RU messages as network-wide broadcasts.

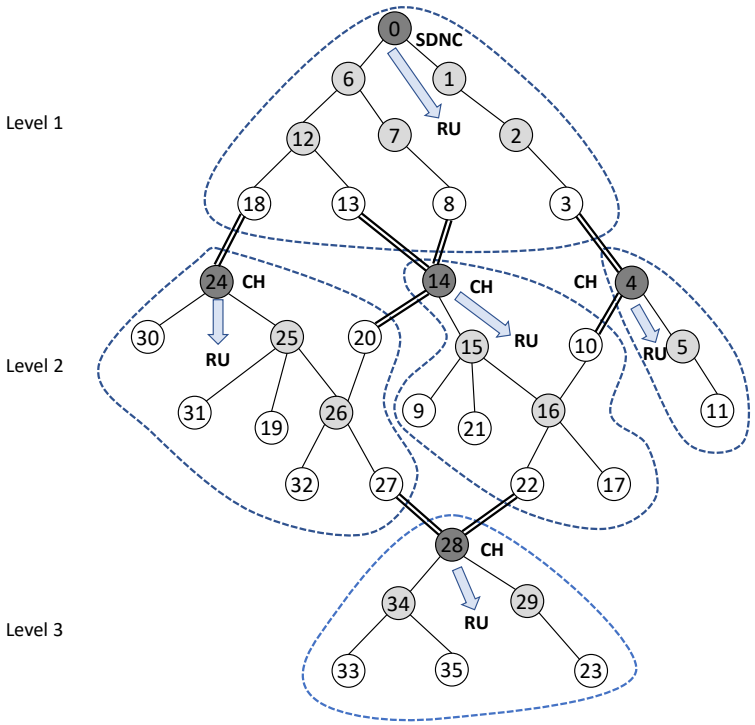


Figure 9.5: Hierarchical view of the clusters formation. SDNC and CHs sending RU messages as network-wide-broadcasts.

9.1.3.2 Inter-Cluster Routing

In addition to the RU messages, the CHs also send the CI messages. However, unlike the RU message, a CH sends the CI message to nodes in another cluster. A CI message includes the cluster topology information, which allows the nodes in another cluster to learn the inter-cluster routes. Each CI message also includes the origin node and a sequence number (for identifying duplicates).

A CH broadcasts the CI message so the gateway nodes in neighboring clusters can receive and forward the CI message to their respective CHs. It always travels from level l to level $\leq l$. We note that nodes do not know their cluster level, nor do they use the cluster level in any way. We use cluster levels only to explain the forwarding process of CI messages.

Similar to the NI message, the nodes send the CI message using the *IA* feature. When a node forwards the CI message, it updates its routes to all nodes included in the message and sets the sender as the next hop - thereby, learning routes for inter-cluster routing. This process results in a hierarchical routing scheme, where nodes in a lower-level cluster (i.e., closer to the SDNC) possibly have more routes than nodes in higher-level clusters.

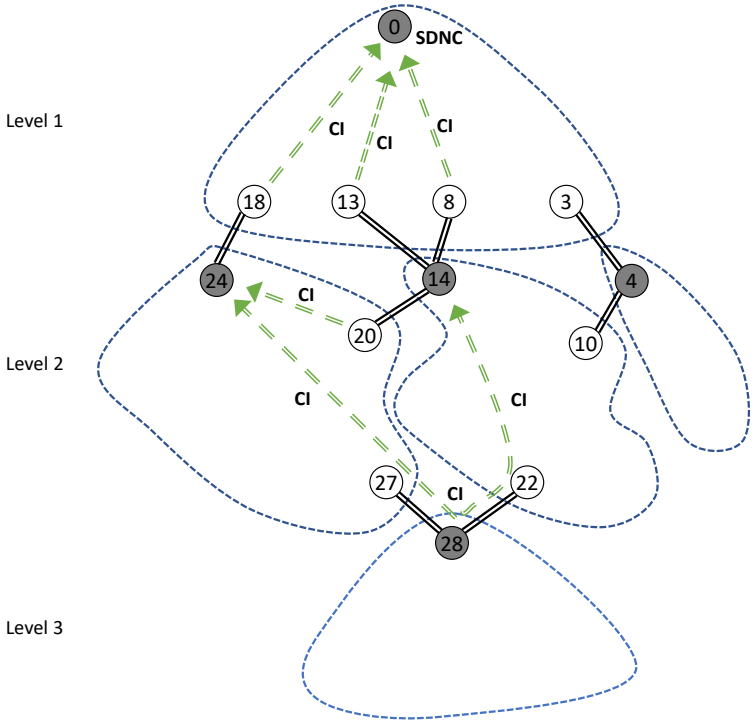


Figure 9.6: CH node 28 sending CI message and gateway nodes forwarding it.

Figure 9.6 shows CH node 28 broadcasting a CI message, which is received by the gateway nodes in the level 2 clusters. These gateway nodes forward the message to their CH via the Route to SDNC (RTS). All intermediate nodes update their routes to all nodes in the cluster of node 28 and set the sender as the next hop. When the message reaches CH nodes 24 and 14, they broadcast it again, so that the gateway

nodes in level 1 (i.e., nodes 8, 13, and 18) can receive and forward it to their CH (i.e., the SDNC). Note that the CI includes a sequence number and the origin node information, which allows nodes to identify the duplicates and forward the message just once.

9.1.3.3 Data packet Forwarding

HCPR being a proactive protocol, all nodes will get periodic intra-cluster route updates. However, a node may or may not have, depending on its cluster level, routes for inter-cluster routing.

Also, nodes invalidate their configured routes on detecting link breaks. A link break is detected towards a neighbor (i.e., the next hop) when the node fails to receive a packet from that neighbor in the past *NbrMaintenance* interval. Nodes may also detect link breaks using the 802.11 CSMA/CA schemes (i.e., RTS/CTS/ACK), if available.

Algorithm 11 Forwarding Data Packets

```
1: procedure ForwardDataPacket(pkt)
2:     if state == critical or state == gateway or node == pkt.source then
3:         if route to pkt.dst is valid then
4:             Unicast pkt to the nextHop
5:         else if RTS is valid then
6:             Unicast pkt to RTS
7:         else
8:             Broadcast pkt
9:         end if
10:    end if
11: end procedure
```

The procedure used for forwarding the data packets is shown in Algorithm 11, and it is similar to the one used by CPR. The only difference is that, in HCRP, along with the critical nodes, the gateway nodes also relay data packets. Similar to CPR, non-critical nodes do not relay data packets. They transmit only if they are the source nodes (the origin nodes). So, if a critical or gateway node receives a data packet for forwarding, it checks for a route to the destination. If the route is available and valid, then the node unicasts the data packet to the next hop. Otherwise, if the RTS is valid,

then the node unicasts to the RTS. In all other cases, the node broadcasts. Note that RTS is the node's next hop in the route to the SDNC.

9.2 Communication Complexity

We now describe HCPR's communication complexity (CC). Similar to the previous chapters, the CC is defined as the total bytes transmitted in the network during its operation. Table 9.1 lists all the symbols used for expressing the CC. For this analysis, we assume that each transmission has a successful delivery.

Table 9.1: Symbols

Category	Symbol	Meaning
Network	**N**	Network Size
	D	Network Diameter
	M	Average Node Degree
	B	Data Packet Size
	N_g	Gateway Nodes
	N_c	Critical Nodes
Messages	H_{td}	Topology Discovery Header Size
	H_{ni}	Neighbor Information Header Size
	H_{ru}	Route Update Header Size
	H_{ci}	Cluster Information Header Size
Protocol	R_{td}	Topology Discovery Rate
	K	Cluster Diameter
Cluster	C_{ch}	Cluster Heads (i.e., Number of Clusters)
	C_c	Average Critical Nodes Per Cluster
	C_s	Average Cluster Size

As described in Section 9.1, HCPR uses four control messages, namely TD, NI, RU, and CI. The communication complexity of each of them is explained in Table 9.2. Equation 9.1 represents the combined control communication complexity of all of them,

Table 9.2: Communication complexity of each message

Message	Complexity	Explanation
TD	NH_{td}	SDNC broadcasts a TD message of size H_{td}, and each node rebroadcasts it.
NI	$(H_{ni} + M)C_{ch}C_cK$	Each critical node in each cluster sends the information of its M neighbors in an NI message to its CH. The cluster radius is K, so up to K nodes may forward.
RU	$(H_{ru} + C_s)C_{ch}C_c^2$	Each CH sends RU messages, one for each of its C_c critical nodes. Each such message includes routes to every node in the cluster (i.e., C_s routes), and it is a network-wide broadcast so forwarded by C_c nodes.
CI	$(H_{ci} + C_s)C_{ch}^2K$	Each CH sends its cluster information (i.e., size C_s) in a CI message. Each such message gets forwarded by up to $C_{ch}K$ nodes, i.e., by all clusters of radius K.

where R_{td} is the topology discovery rate.

$$CC_{ctrl\_hcpr} \leq R_{td}(NH_{td}+(H_{ni}+M)C_{ch}C_cK+(H_{ru}+C_s)C_{ch}C_c^2+(H_{ci}+C_s)C_{ch}^2K) \quad (9.1)$$

For analyzing the Asymptotic Control Communication Complexity (ACCC), we ignore the constants: H_{td}, H_{ni}, H_{ru}, and H_{ci}. If the average node degree is d (i.e., $M = d$), then ACCC is $O(N + dC_{ch}C_cK + C_sC_{ch}C_c^2 + C_sC_{ch}^2K)$.

Note that the total number of critical nodes in all clusters will always be $\leq$ the total number of critical nodes in a non-hierarchical approach like CPR. Thus,

$$C_{ch}C_c \leq N_c \quad (9.2)$$

Also, the total number of nodes in all clusters is equal to the total number of nodes in the network. Thus,

$$C_{ch}C_s = N \quad (9.3)$$

Substituting Equations 9.2 and 9.3 in the ACCC of HCPR, we get $O(N + dN_cK + NC_c^2 + NC_{ch}K)$, which essentially is $O(dN_cK + N(C_c^2 + C_{ch}K))$. Thus,

$$CC_{ctrl\_hcpr} = O(dN_cK + N(C_c^2 + C_{ch}K)) \tag{9.4}$$

Similar to both PCC and CPR, HCPR is also a proactive protocol, so the forwarding tables are periodically updated. Each critical node has a route to every other node in the same cluster but may not have to nodes in other clusters. Assuming nodes send data packets to destinations selected uniformly at random, the probability of sending a packet to one in the same cluster is C_s/N.

Let p_{lb} be the probability of a link break between source and destination. We note that the node broadcasts the data packet if the route becomes invalid due to link break. If the destination is in the same cluster and the route between the source and destination is not valid, then up to $N_c + N_g$ nodes may forward; otherwise, up to 2K (cluster diameter) nodes may forward. Similarly, if the destination is outside the cluster and the route is invalid, then up to $N_c + N_g$ nodes may forward; otherwise, up to D (network diameter) nodes may forward. Equation 9.5 shows the CC of sending a packet.

$$CC_{data\_hcpr} = (1 - p_{lb})B\Big(\frac{C_s}{N}2K + \Big(1 - \frac{C_s}{N}\Big)D\Big) + p_{lb}B(N_c + N_g) \tag{9.5}$$

Considering R_{gen} as the data packet generation rate, then the total communication complexity of HCPR is:

$$CC_{hcpr} = CC_{ctrl\_hcpr} + R_{gen}CC_{data\_hcpr} \tag{9.6}$$

We now analyze HCPR's ACCC for dense and sparse networks. For dense networks, the average node degree d is very high, and the network diameter D is very small, so $d = O(N)$ and $D = O(1)$. Further, a node degree of $O(N)$ results in the selection of only a few critical nodes, hence $N_c = O(1)$ and $C_c = O(1)$. If the

network diameter is very small, then $K = O(1)$, and there will be only a few clusters, so $C_{ch} = O(1)$. Substituting these values in the ACCC in Equation 9.4, we get $O(N+N)$, which is $O(N)$.

For the sparse networks, the node degree is very small, so $d = O(1)$, i.e., d is constant as N increases. In such networks, the number of critical nodes have to grow as $O(N)$ no matter the protocol. Also, D will be $O(N)$. Note that $C_c \leq N_c$, $K \leq D$, and $C_{ch} \leq N_c$ always. Substituting these values in the ACCC in Equation 9.4, we get $O(N^3)$.

Table 9.3: Comparison of the asymptotic control communication complexities

	PCC	**CPR**	**HCPR**
Generic	$O(dND + N^2D)$	$O(dN_cD + NN_c^2)$	$O(dN_cK + N(C_c^2 + C_{ch}K))$
Dense	$O(N^2)$	$O(N)$	$O(N)$
Sparse	$O(N^3)$	$O(N^3)$	$O(N^3)$

Table 9.3 shows a comparison of the asymptotic control communication complexities of PCCP, CPR, and HCPR for generic, dense, and sparse networks. For the dense and sparse networks, the complexities of CPR and HCPR are the same. For the generic networks, the relative gain of HCPR's complexity over CPR depends on the value of K.

If $K \geq D$, which makes HCPR equivalent to CPR, then $C_{ch} = 1$, $C_c = N_c$, and $C_s = N$. Substituting these values in the ACCC of HCPR, we get $CC_{ctrl\_hcpr} \approx CC_{ctrl\_cpr}$. Further, Equation 9.5 becomes the same as Equation 8.3, i.e., $CC_{data\_cpr} = CC_{data\_hcpr}$ because $C_s = N$.

If $K = 0$, which makes each node a cluster in itself, then $C_{ch} = N$, $C_c = 1$, and $C_s = 1$. The CC_{ctrl_hcpr} reduces significantly because complexities of NI and RU become $O(1)$. However, CC_{data_hcpr} increases drastically because $C_s = 1$. So as per

Equation 9.5, a node sends each data packet to a node in another cluster and up to N nodes forward it.

A large value of K makes HCPR perform similar to CPR, whereas a small value reduces the CC of control packets but significantly increases the CC of sending data packets. Thus, K decides the trade-off between the two complexities. In the simulation experiments described in Section 9.3, we have used $K = 3$.

9.3 Simulation Results

We now describe the simulation results of HCPR and compare them to those of OLSR. Table 9.4 lists the simulation scenarios, which are similar to those used for evaluating CPR. We have designed HCPR for large networks, so we evaluate it for networks of size up to 250 nodes. Table 9.5 lists all of the simulation parameters.

Table 9.4: Simulation Scenarios

Network Scenario	**Network Size** (nodes)	**Network Density** (nodes/km^2)	**Data Packet Interval** (seconds)	**Node Speed** (m/s)
Increasing Network Size	[100, 250]	3.3	10	4
Decreasing Network Density	200	[5.5, 2.5]	10	4
Increasing Network Load	200	3.3	[10, 2]	4
Increasing Node Speed	200	3.3	10	[2, 10]

The simulation results are compared using the following three metrics: (1) Packet Delivery Ratio (PDR), which is the ratio of the total data packets received and transmitted, (2) Total Communication Load (TCL), which is the sum of the routing overhead caused by the control packets and size of the data packets, and (3) Average Delay (AD), which is an average end-to-end delay of the data packets. Each result shown is an average of 10 runs.

Similar to the CPR results, we have compared HCPR's results to those of OLSR obtained using Hello intervals: 2, 4, and 6 seconds, and the plots show their results as OLSR-2, OLSR-4, and OLSR-6, respectively. The recommended Hello interval in

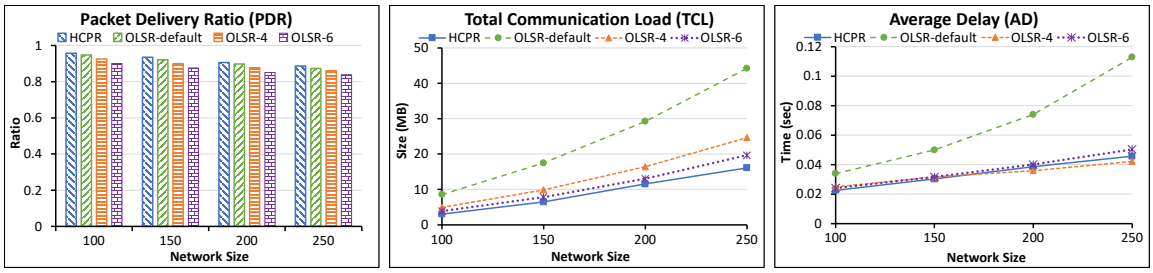
Table 9.5: Simulation Parameters

Parameter	Value	Parameter	Value
Simulation Time	300 seconds	Node Speed	4 m/s
Data Packet Size	200 Bytes	Node Mobility	Rand. Waypoint
Application Nodes	Rand. Src. Dst.	Trans. Power	12 dBm
Propagation Loss	Friis Model	MAC	802.11b
HCPR			
TD Interval	30 secs	NbrMaintenance	Data pkt interval
OLSR			
Hello Interval	[2, 6] seconds	TC Interval	5 seconds

the RFC [35] and ns3 is 2 seconds, but we include results for other intervals as well because they have lower control overhead. A common trend observed in all simulation results is that a larger Hello interval results in reducing the control overhead, but it also reduces the PDR. The larger the interval, the lower is the overhead and the PDR. Intuitively, a larger interval results in an infrequent selection of relay and out-of-sync network topology, causing packet losses and a lower PDR.

We have shown the results of all the Hello intervals in plots, but for brevity, numerical comparisons in the description are included only for OLSR-2.

9.3.1 Increasing Network Size

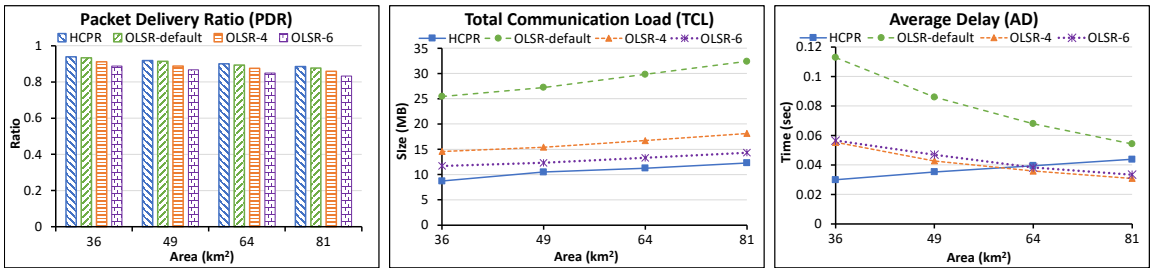


(a) Packet Delivery Ratio (b) Total Communication Load (c) Average Delay

Figure 9.7: Simulation results for scenario 1 (Increasing Network Size) where the network size ranges from 100 to 250 nodes but the density remains constant.

Figure 9.7 shows the results for the increasing network size scenario. The hierarchical routing allows HCPR to attain a better PDR than OLSR-2 for all network sizes (shown in Figure 9.7(a)). If links break due to node mobility, HCPR still maintains a high PDR by leveraging the CDS property of critical nodes. Note that gateway nodes have links to the CHs, so they are also a part of the CDS and help forward data packets for inter-cluster routing. Dividing the network into clusters and using cluster heads for configuring intra-cluster routes significantly reduces the overhead because fewer nodes forward the control message. Figure 9.7(b) shows HCPR having ∼2.7x better (lower) TCL than OLSR-2. A lower TCL also allows HCPR to achieve a high PDR because the interference and packet collisions reduce. Fewer control messages also keep AD low for data packets because of fewer MAC layer back-offs and retransmissions. Figure 9.7(c) shows HCPR having ∼2.4x better (lower) AD than OLSR-2.

9.3.2 Decreasing Network Density



(a) Packet Delivery Ratio (b) Total Communication Load (c) Average Delay

Figure 9.8: Simulation results for scenario 2 (Decreasing Network Density) where network size is 100 nodes but density ranges from approximately 5.5 to 2.5 nodes/km^2 (i.e., simulation area from 36 to 81 km^2)

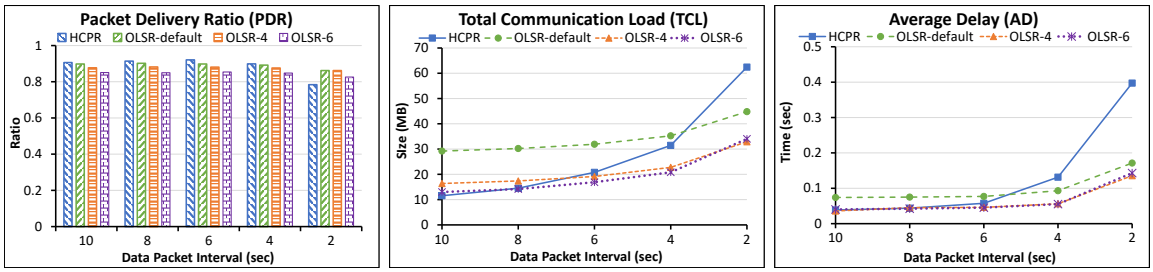
Figure 9.8 shows the results for the decreasing network density scenario. This scenario evaluates the protocols on networks of size 200 nodes, having densities ranging from 5.5 to 2.5 nodes/km^2 (i.e., simulation area between 36 and 81 km^2). HCPR gives a better PDR than OLSR not only in dense networks but also in sparse networks (see Figure 9.8(a)). Dense networks result in the formation of only a few clusters, so nodes

send most data packets to destinations in the same cluster and attain a high PDR. In sparse networks, the inter-cluster routing via the gateway nodes allows HCPR to maintain a high PDR.

Figure 8.2(b) shows increasing TCL with decreasing density because the average distance between the nodes increases so more nodes forward the packets (both control and data). However, HCPR achieves up to ∼2.9x (in dense) and ∼2.6x (in sparse) lower TCL than OLSR-2.

In dense networks, frequent transmission of the large-sized Hello packets results in OLSR-2 having a higher AD for data packets (see Figure 9.8(c)) because nodes experience several MAC layer back-offs. However, as the density decreases, the AD also decreases. HCPR has a contrasting result, in which the decreasing density increases the AD. In sparse networks, hierarchical routing increases the average number of hops between the nodes, and that increases the AD.

9.3.3 Increasing Network Load



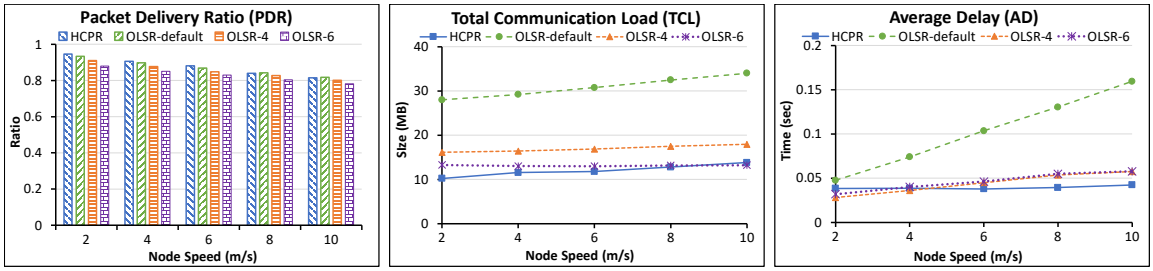
(a) Packet Delivery Ratio (b) Total Communication Load (c) Average Delay

Figure 9.9: Simulation results for scenario 3 (Increasing Network Load) where network size is 200 but data packet interval ranges from 10 to 2 seconds.

Figure 9.9 shows the results for the increasing network load scenario. Figure 9.9(a) shows HCPR having almost the same PDR as OLSR-2 for low loads, but at the maximum load (i.e., an interval of 2 secs) its PDR drops significantly. This drop is the result of nodes transmitting several data packets as broadcast, increasing interference and packet collisions in the network. Figure 9.9(b) confirms that by showing

a steeper increase in the HCPR's TCL over OLSR-2. The increasing broadcast transmissions also increase HCPR's AD. Figure 9.9(c) shows HCPR having a significantly higher AD than OLSR-2. Thus, HCPR gives a better performance than OLSR-2 until average network loads, but for high loads, its PDR starts dropping and TCL and AD increase significantly.

9.3.4 Increasing Node Speed



(a) Packet Delivery Ratio (b) Total Communication Load (c) Average Delay

Figure 9.10: Simulation results for scenario 4 (Increasing Node Speed) where network size is 200 but node speed ranges from 2 to 10 m/s.

Figure 9.10 shows the results for the increasing node speed scenario. Figure 9.10(a) shows PDR decreasing for all with increasing node speeds, but HCPR maintaining a better or similar PDR than OLSR for all node speeds.

Figure 9.10(b) shows TCL increasing for all protocols with increasing node speeds. A higher node speed has contrasting effects on HCPR and OLSR. The Random Waypoint mobility model at high node speeds concentrates more nodes at the center of the simulation area, causing them to be in closer proximity to each other. As a result, HCPR selects fewer critical nodes, and all nodes in OLSR transmits larger-sized Hello packets. In this experiment, both the network size and the network load (traffic) remain the same, so increasing TCL with increasing node speed is a result of an increasing size of Hello packets (in OLSR) and broadcast transmissions due to frequent link breaks (in HCPR). A significant increase in OLSR-2's AD is also a result

of the increasing size of Hello packets. HCPR has the same AD as OLSR-2 at node speed 2 m/s but ∼2.7x lower for node speed 10 m/s (shown in Figure 9.10(c)).

9.4 Conclusions

In this chapter, we presented a hierarchical routing protocol called HCPR designed for our SD-MANET architecture. It addresses scalability issues of the CPR protocol discussed in Section 8.4 by forming clusters in the network. HCPR reduces the communication overhead by allowing Cluster Heads to configure intra-cluster routing and identifying gateway nodes for inter-cluster forwardings.

We have discussed HCPR's communication complexity and shown it to be equal to CPR's in dense and sparse networks, but for the generic case, the gain depends on the cluster radius K. We evaluate HCPR on large-size networks for several scenarios and show its improvement over OLSR – and hence, over CPR. HCPR's better performance than CPR is a result of its hierarchical routing scheme and lower communication complexity, which keeps the interference to a minimum and reduces packet losses.

Chapter 10

SUMMARY AND FUTURE DIRECTIONS

Software-Defined Networking (SDN) has brought about a paradigm shift in the way networks are designed. Although proposed originally for wired and data center networks, a large number of the wireless domains have adopted and benefited from the SDN architecture.

We have identified several benefits and opportunities that the SDN architecture can facilitate in Mobile Ad Hoc Networks (MANETs). However, using an SDN-based architecture for managing MANETs has been challenging, mainly due to the dynamic nature of the network topology. There have been several SDN-based architectures proposed for MANETs in the past few years. However, most, if not all, of these architectures are inadequate for infrastructure-less MANETs.

MANETs characterized by low rates and long ranges often have ultra-low capacities (i.e., network bandwidth). In most situations, the capacity is so low that control packets used by the routing protocols themselves occupy a majority of the available bandwidth and overwhelm the network. Thus, all existing routing protocols are inadequate in such ultra-low capacity MANETs.

In this dissertation, we have presented an SDN-based architecture suitable for infrastructure-less MANETs. We have designed several centralized routing protocols for our proposed architecture. These protocols cater to the needs for reactive, proactive, and hierarchical routing strategies needed in MANET. We have also designed two zero-control-packet routing protocols for addressing the challenges of ultra-low capacity MANETs.

We first summarize the work presented in this dissertation and highlight our contributions. Later, we suggest directions for future research. In the end, we give a

brief overview of other research projects conducted by us that are not a part of this dissertation but have led to publications.

10.1 Summary

We have designed an architecture [45, 85] for Software-Defined Mobile Ad Hoc Network (SD-MANET) that has none of the following constraints: (1) infrastructure for hosting the SDNC, (2) single-hop (direct) out-of-band control communication links between the SDNC and each node, (3) location services for tracking the position of nodes and learning the network topology, or (4) preexisting IP connectivity for control communication. Because our SD-MANET architecture is not limited by the above constraints, it is suitable for infrastructure-less networks having low-capacity links and susceptibility to high interference, collisions, and packet losses.

We have recognized three functions necessary for managing the network from a centralized location, i.e., the SDN Controller (SDNC). These functions are (1) learning route to SDNC, (2) learning network topology, and (3) sending network routes.

Using the above three SD-MANET functions, we have designed a proactive routing protocol called PCC. We have shown that the PCC protocol attains a better delivery ratio and a lower communication overhead than both DSDV and OLSR routing protocols for networks of size up to 50 nodes.

For MANETs characterized by ultra-low capacities, we have proposed an architecture [98] and designed two zero-control-packet protocols: ECHO [47] and VINE [48]. These protocols do not use any control packets whatsoever. Instead, they include some additional information in the data packet header that sets the states in the nodes for forwarding data packets. The additional information remains constant in size and does not scale with network size or density. Moreover, having this information in the data packet header also prevents per-packet MAC- and PHY-layer header and MAC contention penalties.

We have designed the ECHO protocol to perform efficient network-wide broadcasts. We select nodes in the network (called critical nodes) that form a Connected

Dominating Set (CDS) of the network graph. We have formally proved that transmissions of the selected critical nodes are sufficient for a source-independent network-wide broadcast. Unlike some of the other network-wide broadcasting schemes, ECHO is deterministic, source-independent, mobility-accommodating, and balancing the battery consumption across nodes. We have shown using simulations that ECHO significantly outperforms both Flooding and Multi-Point Relay (MPR) [94] with up to 20% improvement in the packet delivery ratio and up to 4x less communication load. We have also shown that for dense networks, the asymptotic communication complexity of ECHO is $O(N)$ lower than that of Flooding and MPR.

We have designed the VINE protocol using a gradient-based routing scheme that delivers a packet to the destination specified in the header. VINE delivers packets reliably using features such as Implicit Acknowledgments and End-to-End Acknowledgments. We have shown using simulations that VINE significantly outperforms AODV by providing up to 2.5x higher delivery ratio and up to 1.2x less communication load.

We have optimized the SD-MANET functions by employing some of the features of the ECHO and VINE protocols. In particular, we select critical nodes using the ECHO protocol and allow the SDNC to learn the network topology using the neighborhood information of only these critical nodes. Further, the SDNC efficiently disseminates the routing information using critical nodes as network-wide broadcasts. We have also used the Implicit Acknowledgment feature of the VINE protocol for improving the per-hop reliability of control messages. The use of critical nodes for learning network topology and disseminating routing information reduces the communication overhead. The use of Implicit Acknowledgment for sending control messages allows the SDNC to learn the network topology reliably.

Using these optimized SD-MANET features, we have designed three centralized routing protocols: CORR, CPR, and HCPR.

We have designed CORR to be a reactive protocol, in which nodes send messages for requesting routes from the SDNC. On receiving a request, the SDNC opportunistically updates routes in all critical nodes, making them the network backbone for

forwarding data packets. We have shown using simulations that CORR outperforms AODV by providing up to 10% better packet delivery ratio, 1.5x less communication load, and 3x less delay.

The CPR protocol is designed to be proactive. The SDNC periodically updates routing information in all nodes in the network. We have proved using theoretical analysis that CPR has O(N) lower communication complexity than PCC in dense networks. We have also shown using simulations that CPR provides the same or better delivery ratio than OLSR but causes up to 2.4x less communication load and up to 1.4x lower delay for networks of size 100 nodes, making CPR a better (scalable) protocol than PCC.

We have improved the scalability of CPR further by designing a hierarchical protocol called HCPR. The HCPR protocol builds clusters in the network and configures inter- and intra-cluster routing. We have shown using theoretical analysis that HCPR's communication complexity gain over CPR depends on the value of cluster radius (K). We have shown using simulations that HCPR provides better or same delivery ratio than OSLR but causes up to 2.7x lower communication load and up to 2.4x lower delay for networks of size up to 250 nodes.

All the above features improve the control communication in SD-MANET routing protocols by either reducing communication load or improving reliability. But the feature that gives an edge to our protocols over most traditional routing protocols is the way nodes forward data packets. The intrinsic dynamic nature of MANETs continues to cause link breaks, invalidate the configured routes, and disrupt the ongoing communication. Traditional routing protocols use different schemes for detecting link breaks and then updating routes in the affected nodes. However, during this route update process, nodes either drop the data packets or buffer them, resulting in either low delivery ratio or high delay. By contrast, when a link breaks, instead of dropping or buffering, our routing protocols broadcast the data packets. Thus, in our routing protocols, the data packet forwarding is a combination of unicast and broadcast. Nodes

leverage the CDS property of critical nodes for delivering data packets to their destinations. Although this feature increases the data communication load for the broadcasted packets, it results in reducing the control communication load, attaining a high packet delivery ratio, and lowering the delay. Further, the SDNC being a part of the CDS identifies the broadcast transmission and opportunistically updates routes in nodes, reducing the need for broadcasting subsequent data packets.

Below we summarize all the contributions of this dissertation.

1. Most existing SDN-based architectures for MANETs are for infrastructure-based networks, but we have designed an SD-MANET architecture that is suitable for infrastructure-less MANETs. We have designed several centralized routing protocols that are suitable for our SD-MANET architecture.

2. We have presented theoretical analyses for all our routing protocols. We have also evaluated them in an enhanced ns-3 simulation framework and conducted detailed performance evaluation studies. We have designed a wide range of scenarios for our evaluation studies, taking into consideration the network size, network density, node mobility, and traffic load, and compared the results to those of the state-of-the-art MANET routing protocols. Our centralized routing protocols break the dogma that the centralized approaches are inappropriate and unscalable for MANETs. We have shown that not only our centralized routing protocols are competitive in performance to the state-of-the-art decentralized routing protocols but also better than them in most scenarios.

3. We have designed our architecture and routing protocols to be generic enough to apply to any centralized architecture for mesh, sensor, ad hoc, vehicular, and IoT networks.

4. Our zero-control-packet routing protocols presented a radical departure from the prevalent thinking that routing requires collecting topology information via control packets. We prove the significance of these protocols not only through simulations but also through their deployment in the goTenna Pro mesh devices [4]. These devices are being successfully used in fighting forest fires, the aftermath of hurricanes, and military operations [19].

10.2 Future Work

This section presents several future research directions for extending our work on SD-MANET.

SDNC Failure

Every centralized architecture is vulnerable to a single point of failure in the system. Our SD-MANET architecture is no different. If the SDNC fails, then all nodes will stop receiving route updates, making the network incapable of functioning. The SDN-based wired networks address this issue by deploying the SDNC on a cluster, but such solutions may not be practicable for MANETs.

A multi-SDNC architecture can overcome the single point of failure but would need communication between the SDNCs for state sharing and decision-making. Another approach could be to design an SDNC election algorithm. In situations where nodes fail to receive any communication from the SDNC, they can initiate an election procedure for electing a new SDNC. This approach could also address our assumption of having a preselected SDNC in the network. On network startup, an election algorithm can identify and select one of the mobile nodes to become the SDNC. Further, in situations where node mobility results in isolating nodes and making the network disconnected, the isolated nodes can elect an SDNC and form another network. In addition to the splitting scenarios, the election algorithms also need to address the merging scenarios, in which disconnected networks merge over time.

Link Prediction

The SDNC learns the network topology using the local connectivity information of each node. Despite using reliability schemes, a few control packets may fail to reach the SDNC. As a result, the SDNC may not know the local connectivity information of a few nodes, and hence, may not know the complete topology and fail to select optimal routes. To remedy this, the SDNC can employ link prediction techniques [124, 78] and include high probability links in the topology before selecting the routes. Over the years, there have been several heuristics proposed for predicting missing links in network graphs. They are (1) Common Neighbors (CN) [120], (2) Preferential Attachment (PA) [27], (3) Adamic-Adar (AA) [20], and (4) Resource Allocation (RA) [126]. The

SDNC can employ either of them and use a non-deterministic network topology for selecting routes, and possibly reduce the rate of collecting the connectivity information, and hence, the network load.

Dynamic Adjustment of Routing Parameter Values

Every routing protocol uses parameters, such as Hello intervals and expiration time, and preconfigures their value based on the network characteristics. Typically, changing these values would require reconfiguring the nodes offline. A centralized network architecture presents the ability to use centralized algorithms for determining the optimal values based on the network's global view and configuring them via control messages. Dynamically updating the parameter values can help to improve the network performance by reducing the overhead, increasing the delivery ratio, or both.

Multiple Radios

Nodes provisioned with multiple radios can configure them on different frequencies or use a different RF technology for each of them. Our architecture could be extended to use separate radios for control and data communications, preventing one from interfering with the other. However, the RF technology used in the radio essentially determines the transmission range (i.e., the connectivity), and hence, the network topology. Using different RF technologies may result in SDNC learning and maintaining two different topologies and configures routes in both of them.

Cognitive Radios

Cognitive radios are best suited for centralized architectures because the SDNC can run one of the several topology control algorithms [104] proposed over the years for determining the transmission powers. These algorithms can use the SDNC's global network view and the available transmission power options for selecting the optimal value for the current situation and try solving the near-far problem. [86]. However, such solutions may require relaxing requirements such as multi-hop communication between the SDNC and the nodes.

10.3 Other Research Projects

Besides the work on SD-MANET, some of our other research projects that have resulted in publications are (1) a comprehensive literature survey on fault detection and localization techniques for computer networks [50], (2) techniques for generating probes for monitoring and diagnosing faults in large-scale networks [51], and (3) improving network utilization of Docker containers [52, 62].

The literature survey [50] has fault localization techniques classified, based on their applicability and key design principles, into five categories: (1) active monitoring techniques, (2) techniques for overlay and virtual networks, (3) decentralized probabilistic management techniques, (4) temporal correlation techniques, and (5) learning techniques.

The probe generation techniques [51] use several heuristics and network partitioning strategies for identifying candidate probes that result in the selection of efficient target probes for monitoring and diagnosing faults in large networks.

Docker's best-effort networking and other resources, such as I/O and CPU, were enhanced to include QoS functionalities [52, 62]. Priorities were assigned to containers using queuing disciplines for shaping the ingress and egress traffic.

BIBLIOGRAPHY

[1] DASH7. http://www.dash7-alliance.org. Accessed: Aug-2019.

[2] EC-GSM-IoT. https://www.gsma.com/iot/wp-content/uploads/2016/10/3GPP-Low-Power-Wide-Area-Technologies-GSMA-White-Paper.pdf. Accessed: Aug-2019.

[3] goTenna Mesh. https://gotenna.com/. Accessed: Aug-2019.

[4] goTenna Pro. https://www.gotenna.com/pages/gotenna-pro-homepage. Accessed: Nov-2018.

[5] I-Scoop. https://www.i-scoop.eu/internet-of-things-guide/iot-network-lora-lorawan/. Accessed: Aug-2019.

[6] IEEE 802.15 WPAN Task Group 4 (TG4). http://www.ieee802.org/15/pub/TG4.html. Accessed: Aug-2019.

[7] INGENU. https://www.ingenu.com/technology/rpma/. Accessed: Aug-2019.

[8] IQRF. https://www.iqrf.org/technology/. Accessed: Aug-2019.

[9] Link Labs. https://www.link-labs.com/blog/what-is-sigfox. Accessed: Aug-2019.

[10] LoRaWAN. https://lora-alliance.org/. Accessed: Aug-2019.

[11] LTE Cat M1. https://www.u-blox.com/en/lte-cat-m1-old. Accessed: Aug-2019.

[12] Next Generation Mobile Networks (NGMN). https://www.ngmn.org/home.html. Accessed: Aug-2019.

[13] ns-3.25. https://www.nsnam.org/.

[14] OpenFlow Specification 1.0. https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf. Accessed: Aug-2019.

[15] Sigfox. https://www.sigfox.com/. Accessed: Aug-2019.

[16] Weightless. http://www.weightless.org. Accessed: Aug-2019.

[17] Zigbee. https://www.zigbee.org/. Accessed: Aug-2019.

[18] North Atlantic Treaty Organization (NATO) Standardization Agreement (STANAG) 5631/AComP-5631, Narrowband Waveform Physical Layer, Ratification Draft, Edition 1, 2015.

[19] goTenna Deployment After Action Reports. https://gotennapro.com/pages/resources#case-studies, 2019. Accessed: 2019-08-01.

[20] Lada A Adamic and Eytan Adar. Friends and neighbors on the Web. *Social Networks*, 25(3):211–230, Jul. 2003.

[21] Ferran Adelantado, Xavier Vilajosana, Pere Tuset-Peiro, Borja Martinez, Joan Melia-Segui, and Thomas Watteyne. Understanding the Limits of LoRaWAN. *IEEE Communications Magazine*, 55(9):34–40, 2017.

[22] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[23] Ian F. Akyildiz, Xudong Wang, and Weilin Wang. Wireless mesh networks: a survey. *Computer Networks*, 47(4):445–487, 2005.

[24] Muhammad Aslam, Xiaopeng Hu, and Fan Wang. SACFIR: SDN-Based Application-Aware Centralized Adaptive Flow Iterative Reconfiguring Routing Protocol for WSNs. *Sensors*, 17(12):2893, Dec. 2017.

[25] Aloÿs Augustin, Jiazi Yi, Thomas Clausen, and William Mark Townsley. A Study of LoRa: Long Range & Low Power Networks for the Internet of Things. *Sensors*, 16(9), 2016.

[26] Emmanuel Baccelli, Matthias Philipp, and Mukul Goyal. The P2P-RPL routing protocol for IPv6 sensor networks: Testbed experiments. In *Proc. of the 19th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2011*, pages 1–6, Split, Croatia, Sep. 2011.

[27] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, Oct. 1999.

[28] Jean-Paul Bardyn, Thierry Melly, Olivier Seller, and Nicolas Sornin. IoT: The era of LPWAN is starting now. In *Proc. of the ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, pages 25–30, Lausanne, Switzerland, Sep. 2016.

[29] Paolo Bellavista, Alessandro Dolci, and Carlo Giannelli. MANET-oriented SDN: Motivations, Challenges, and a Solution Prototype. In *Proc. of the 2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 14–22, Chania, Greece, Jun. 2018.

[30] Martin C. Bor, Utz Roedig, Thiemo Voigt, and Juan M. Alonso. Do LoRa Low-Power Wide-Area Networks Scale? In *Proc. of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 59–67, Malta, Malta, Nov. 2016.

[31] Azzedine Boukerche, Begumhan Turgut, Nevin Aydin, Mohammad Z. Ahmad, Ladislau Bölöni, and Damla Turgut. Routing protocols in ad hoc networks: A survey. *Computer Networks*, 55(13):3032–3080, 2011.

[32] Gruia Călinescu, Ion I Măndoiu, Peng-Jun Wan, and Alexander Z Zelikovsky. Selecting forwarding neighbors in wireless ad hoc networks. *Mobile Networks and Applications*, 9(2):101–111, 2004.

[33] Julien Cartigny and David Simplot. Border node retransmission based probabilistic broadcast protocols in ad-hoc networks. In *Proc. of the 36th Annual Hawaii International Conference on System Sciences, 2003*, Big Island, HI, Jan. 2003.

[34] Marco Cattani, Carlo Alberto Boano, and Kay Uwe Römer. An Experimental Evaluation of the Reliability of LoRa Long-Range Low-Power Wireless Communication. *Journal of Sensor and Actuator Networks*, 6(2), Jun. 2017.

[35] T. H. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). https://tools.ietf.org/html/rfc3626, Oct. 2003. RFC 3626.

[36] Thomas Clausen, Christopher Dearlove, Justin Dean, and Cedric Adjih. Generalized Mobile Ad Hoc Network (MANET) Packet/Message Format. RFC 5444, Feb. 2009.

[37] Sergio Correia, Azzedine Boukerche, and Rodolfo I. Meneguette. An architecture for hierarchical software-defined vehicular networks. *IEEE Communications Magazine*, 55(7):80–86, 2017.

[38] Alejandro De Gante, Mohamed Aslan, and Ashraf Matrawy. Smart wireless sensor network management based on software-defined networking. In *Proc. of the 27th Biennial Symposium on Communications, QBSC*, pages 71–75, Kingston, ON, Canada, Jun. 2014.

[39] Eli De Poorter, Jeroen Hoebeke, Matthias Strobbe, Ingrid Moerman, Steven Latré, Maarten Weyn, Bart Lannoo, and Jeroen Famaey. Sub-GHz LPWAN Network Coexistence, Management and Virtualization: An Overview and Open Research Challenges. *Wireless Personal Communications*, 95(1):187–213, Jul 2017.

[40] Peter Dely, Andreas Kassler, and Nico Bayer. OpenFlow for Wireless Mesh Networks. In *Proc. of the 20th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6, Maui, USA, Jul. 2011.

[41] Andrea Detti, Claudio Pisa, Stefano Salsano, and Nicola Blefari-Melazzi. Wireless Mesh Software Defined Networks (wmSDN). In *Proc. of the 2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 89–95, Lyon, France, Oct. 2013.

[42] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.

[43] Avri Doria, Jamal Hadi Salim, Robert Haas, Hormuzd Khosravi, Weiming Wang, Ligang Dong, Ram Gopal, and Joel Halpern. Forwarding and Control Element Separation (ForCES) Protocol Specification. RFC 5810, Mar. 2010.

[44] Bharat Doshi and Derya Cansevar. Software Defined Networking for Army's Tactical Network: Promises, Challenges, Architectural Approach, and Required S & T Work. 2016.

[45] Ayush Dusia, Vinod K. Mishra, and Adarshpal S. Sethi. Control Communication in SDN-based Dynamic Multi-hop Wireless Infrastructure-less Networks. In *Proc. of the 2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, Dec. 2018.

[46] Ayush Dusia, Ram Ramanathan, Vinod K. Mishra, and Adarshpal S. Sethi. Centralized Routing Protocols for Mobile Ad Hoc Networks. [In preparation for submission to the Ad Hoc Networks journal].

[47] Ayush Dusia, Ram Ramanathan, Warren Ramanathan, Christophe Servaes, and Adarshpal S. Sethi. ECHO: Efficient Zero-Control-Packet Broadcasting for Mobile Ad Hoc Networks. *IEEE Transactions on Mobile Computing*. [Submitted].

[48] Ayush Dusia, Ram Ramanathan, Warren Ramanathan, Christophe Servaes, and Adarshpal S. Sethi. VINE: Zero-Control-Packet Routing for Ultra-Low-Capacity Mobile Ad Hoc Networks. In *Proc. of the 2019 IEEE Military Communications Conference (MILCOM)*, Nov. 2019.

[49] Ayush Dusia, Ram Ramanathan, and Adarshpal S. Sethi. CORR: Centralized Opportunistic Reactive Routing for Mobile Multi-hop Wireless Networks. In *Proc. of the 28th International Conference on Computer Communications and Networks (ICCCN 2019)*, Jul. 2019.

[50] Ayush Dusia and Adarshpal S. Sethi. Recent advances in fault localization in computer networks. *IEEE Communications Surveys and Tutorials*, 18(4):3030–3051, 2016.

[51] Ayush Dusia and Adarshpal S. Sethi. Probe generation for active probing. *International Journal of Network Management*, 28(4), May 2018.

[52] Ayush Dusia, Yang Yang, and Michela Taufer. Network Quality of Service in Docker Containers. In *Proc. of the 2015 IEEE International Conference on Cluster Computing*, pages 527–528, Chicago, IL, USA, Sep. 2015.

[53] Jakob Eriksson, Michalis Faloutsos, and Srikanth Krishnamurthy. Routing Scalability in MANETs. In Jie Wu, editor, *Handbook On Theoretical And Algorithmic Aspects Of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks*, chapter 1. Auerbach Publications, Boston, MA, USA, 2005.

[54] Elias C. Eze, Sijing Zhang, and Enjie Liu. Vehicular ad hoc networks (VANETs): Current state, challenges, potentials and way forward. In *Proc. of the 20th International Conference on Automation and Computing*, pages 176–181, Philadelphia, PA, Sep. 2014.

[55] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin. Highly-Resilient, Energy-Efficient Multipath Routing in Wireless Sensor Networks. *Mob. Comput. Commun. Rev.*, 5(4):11–25, Oct. 2001.

[56] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[57] Orestis Georgiou and Usman Raza. Low Power Wide Area Network Analysis: Can LoRa Scale? *IEEE Wireless Communications Letters*, 6(2):162–165, Apr. 2017.

[58] Sudipto Guha and Samir Khuller. Approximation Algorithms for Connected Dominating Sets. *Algorithmica*, 20(4):374–387, Apr. 1998.

[59] Akram Hakiri, Aniruddha Gokhale, Pascal Berthou, Douglas Schmidt, and Thierry Gayraud. Software-Defined Networking: Challenges and Research Opportunities for Future Internet. *Computer Networks*, 75:453–471, Dec. 2014.

[60] Israat Tanzeena Haque and Nael Abu-Ghazaleh. Wireless Software Defined Networking: A Survey and Taxonomy. *IEEE Communications Surveys Tutorials*, 18(4):2713–2737, Fourthquarter 2016.

[61] Zongjian He, Jiannong Cao, and Xuefeng Liu. SDVN: Enabling rapid network innovation for heterogeneous vehicular communication. *IEEE Network*, 30(4):10–15, Jul 2016.

[62] Stephen Herbein, Ayush Dusia, Aaron Landwehr, Sean McDaniel, Jose Monsalve, Yang Yang, Seetharami R. Seelam, and Michela Taufer. Resource management for running hpc applications in container clouds. In Julian M. Kunkel, Pavan Balaji, and Jack Dongarra, editors, *High Performance Computing*, pages 261–278, Cham, 2016. Springer International Publishing.

[63] Guido R Hiertz et al. IEEE 802.11s: The WLAN mesh standard. *IEEE Wireless Communications*, 17(1):104–111, 2010.

[64] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proc. Int. Conf. Mobile Comput. Netw.*, pages 56–67, Boston, Massachusetts, USA, Aug. 2000.

[65] S. M. Riazul Islam, Daehan Kwak, MD. Humaun Kabir, Mahmud Hossain, and Kyung-Sup Kwak. The Internet of Things for Health Care: A Comprehensive Survey. *IEEE Access*, 3:678–708, Jun. 2015.

[66] Dali Ismail, Mahbubur Rahman, and Abusayeed Saifullah. Low-power Wide-area Networks: Opportunities, Challenges, and Directions. In *Proc. of the Workshop Program of the 19th International Conference on Distributed Computing and Networking, ICDCN 2018*, pages 8:1–8:6, Varanasi, India, Jan. 2018.

[67] Atsushi Iwata, Ching-Chuan Chiang, Guangyu Pei, Mario Gerla, and Tsu-Wei Chen. Scalable routing strategies for ad hoc wireless networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1369–1379, Aug. 1999.

[68] Philippe Jacquet, Anis Laouiti, Pascale Minet, and Laurent Viennot. Performance of Multipoint Relaying in Ad Hoc Mobile Routing Protocols. In *Proc. of the Second International IFIP-TC6 Networking Conference on Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; and Mobile and Wireless Communications*, pages 387–398, Pisa, Italy, May 2002.

[69] Philippe Jacquet, Paul Muhlethaler, Thomas Clausen, Anis Laouiti, Amir Qayyum, and Laurent Viennot. Optimized link state routing protocol for ad hoc networks. In *Proc. of the IEEE International Multi Topic Conference, INMIC 2001*, pages 62–68, Lahore, Pakistan, Dec. 2001.

[70] David B. Johnson, David A. Maltz, and Josh Broch. Ad Hoc Networking. chapter DSR: The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks, pages 139–172. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[71] Young-Bae Ko and Nitin Vaidya. Geotora: a protocol for geocasting in mobile ad hoc networks. In *Proc. of the International Conference on Network Protocols*, pages 240–250, Osaka, Japan, Nov. 2000.

[72] Young-Bae Ko and Nitin H. Vaidya. Location-aided Routing (LAR) in Mobile Ad Hoc Networks. In *Proc. of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, MobiCom 1998, pages 66–75, Dallas, Texas, 1998.

[73] Ian Ku, You Lu, and Mario Gerla. Software-Defined Mobile Cloud: Architecture, services and use cases. In *Proc. of 10th International Wireless Communications and Mobile Computing Conference*, pages 1–6, Nicosia, Cyprus, Aug. 2014.

[74] Ian Ku, You Lu, Mario Gerla, Rafael L. Gomes, Francesco Ongaro, and Eduardo Cerqueira. Towards software-defined VANET: Architecture and services. In *Proc. of the 2014 13th Annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET)*, pages 103–110, Piran, Slovenia, Jun. 2014.

[75] Ian Ku, You Lu, Mario Gerla, Rafael L. Gomes, Francesco Ongaro, and Eduardo Cerqueira. Towards software-defined VANET: Architecture and services. In *Proc. of the 13th Annual Mediterranean Ad Hoc Networking Workshop, MED-HOC-NET 2014*, pages 103–110, Piran, Slovenia, Jun 2014.

[76] Terje Lassen. White paper: Long-range RF communication: Why narrowband is the de facto standard. http://www.mouser.com/pdfdocs/TI-Long-range-RF-communication.pdf, March 2014. Accessed: Aug-2019.

[77] Li Li, Humphrey Rutagemwa, J. Hu ; Phil Hugg, Phil Vigneron, Colin Brown, and Thomas Kunz. Networking for next generation NBWF radios. In *Proc. of the 2015 IEEE Military Communications Conference*, pages 121–126, Tampa, FL, Oct. 2015.

[78] David Liben-Nowell and Jon Kleinberg. The Link Prediction Problem for Social Networks. In *Proc. of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, pages 556–559, New Orleans, LA, USA, Nov. 2003.

[79] Hyojun Lim and Chongkwon Kim. Multicast tree construction and flooding in wireless ad hoc networks. In *Proc. ACM MSWIM*, pages 61–68, 2000.

[80] Hyojun Lim and Chongkwon Kim. Flooding in wireless ad hoc networks. *Computer Communications*, 24(3-4):353–363, 2001.

[81] D. Lundell, A. Hedberg, C. Nyberg, and E. Fitzgerald. A Routing Protocol for LoRA Mesh Networks. In *Proc. of the 2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 14–19, Chania, Greece, Jun. 2018.

[82] Victoria Manfredi, Ram Ramanathan, Will Tetteh, Regina Hain, and Dorene Ryder. SHARE: Scalable Hybrid Adaptive Routing for dynamic multi-hop Environments. In *Proc. IEEE Conf. on Ubiquitous Intelligence and Computing*, pages 1–8, San Francisco, CA, USA, Aug. 2017.

[83] Paul J. Marcelis, Vijay S. Rao, and R. Venkatesha Prasad. DaRe: Data Recovery through Application Layer Coding for LoRaWAN. In *Proc. of the 2017*

IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI), pages 97–108, Pittsburgh, PA, Apr. 2017.

[84] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks Nick. *ACM SIGCOMM Computer Communication Review*, 38(2):69, Mar 2008.

[85] Vinod K. Mishra, Ayush Dusia, and Adarshpal S. Sethi. Routing in Software-Defined Mobile Ad hoc Networks (SD-MANET). Technical Report ARL-TR-8469, US Army Research Laboratory, Aug. 2018.

[86] Alaa Muqattash, Marwan Krunz, and William E. Ryan. Solving the near–far problem in CDMA-based ad hoc networks. *Ad Hoc Networks*, 1(4):435–453, Nov. 2003.

[87] Katia Obraczka, Kumar Viswanath, and Gene Tsudik. Flooding for reliable multicast in multi-hop ad hoc networks. *Wireless networks*, 7(6):627–634, 2001.

[88] Guangyu Pei, Mario Gerla, and Tsu-Wei Chen. Fisheye state routing: a routing scheme for ad hoc wireless networks. In *Proc. of the 2000 IEEE International Conference on Communications*, pages 70–74, New Orleans, LA, Apr. 2000.

[89] Wei Peng and Xi-Cheng Lu. On the reduction of broadcast redundancy in mobile ad hoc networks. In *Proc. of the 2000 First Annual Workshop on Mobile and Ad Hoc Networking and Computing. MobiHOC*, pages 129–130, Boston, MA, Aug. 2000.

[90] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. Ad hoc On-Demand Distance Vector (AODV) Routing. https://tools.ietf.org/html/rfc3561, Jul. 2003. RFC 3561.

[91] Charles E. Perkins and Pravin Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *Proc. of the Conference on Communications Architectures, Protocols and Applications*, SIGCOMM '94, pages 234–244, London, United Kingdom, 1994.

[92] Charles E. Perkins and Elizabeth Royer. Ad hoc On-Demand Distance Vector Routing. In *Proc. 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, New Orleans, LA, USA, Feb. 1999.

[93] Amin Qayyum, Laurent Viennot, and Anis Laouiti. Multipoint relaying for flooding broadcast messages in mobile wireless networks. In *Proc. of the 35th Annual Hawaii International Conference on System Sciences*, pages 3866–3875, Big Island, HI, Jan. 2002.

[94] Amin Qayyum, Laurent Viennot, and Anis Laouiti. Multipoint Relaying: An Efficient Technique for Flooding in Mobile Wireless Networks. Technical Report RR-3898, INRIA, May 2006.

[95] Ram Ramanathan. Whitepaper: On the connectivity of mesh networks. https://inthemesh.com/archive/whitepaper-connectivity-of-mesh-networks/, May 2018. Accessed: Aug 2019.

[96] Ram Ramanathan. Long-Range Short-Burst Mobile Mesh Networking: Requirements, Challenges And Solutions. https://inthemesh.com/archive/long-range-short-burst-mobile-mesh-networking/, Jun. 2019. Accessed: Aug 2019.

[97] Ram Ramanathan, Christophe Servaes, and Warren Ramanathan. ECHO: Efficient Zero-Control Network-Wide Broadcast for Mobile Multi-Hop Wireless Networks. In *Proc. of the 2018 IEEE Military Communications Conference (MILCOM)*, Oct. 2018.

[98] Ram Ramanathan, Christophe Servaes, Warren Ramanathan, Ayush Dusia, and Adarshpal S. Sethi. Long-Range Short-Burst Mobile Mesh Networking: Architecture and Evaluation. In *Proc. of the 2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, Jun. 2019.

[99] Wei Ren, Qing Zhao, Ram Ramanathan, Jianhang Gao, Ananthram Swami, Amotz Bar-Noy, Matthew P. Johnson, and Prithwish Basu. Broadcasting in Multi-radio Multi-channel Wireless Networks Using Simplicial Complexes. *Wireless Networks*, 19(6):1121–1133, Aug. 2013.

[100] Robert D. Poor. Gradient Routing in Ad Hoc Networks. https://www.media.mit.edu/pia/Research/ESP/texts/poorieeepaper.pdf, 2000. MIT Media Laboratory.

[101] Abusayeed Saifullah, Mahbubur Rahman, Dali Ismail, Chenyang Lu, Ranveer Chandra, and Jie Liu. SNOW: Sensor Network over White Spaces. In *Proc. of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, SenSys '16, pages 272–285, Stanford, CA, USA, 2016.

[102] Abusayeed Saifullah, Mahbubur Rahman, Dali Ismail, Chenyang Lu, Jie Liu, and Ranveer Chandra. Enabling Reliable, Asynchronous, and Bidirectional Communication in Sensor Networks over White Spaces. In *Proc. of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, pages 9:1–9:14, Delft, Netherlands, 2017.

[103] Prince Samar, Marc R. Pearlman, and Zygmunt. J. Haas. Independent zone routing: an adaptive hybrid routing framework for ad hoc wireless networks. *IEEE/ACM Transactions on Networking*, 12(4):595–608, Aug. 2004.

[104] Paolo Santi. Topology control in wireless ad hoc and sensor networks. *ACM Computing Surveys*, 37(2):164–194, Jun. 2005.

[105] Yoav Sasson, David Cavin, and André Schiper. Probabilistic broadcast for flooding in wireless mobile ad hoc networks. In *Proc. of the 2003 IEEE Wireless Communications and Networking, 2003. WCNC 2003.*, pages 1124–1130, New Orleans, LA, Mar 2003.

[106] Prasun Sinha, Raghupathy Sivakumar, and Vaduvur Bharghavan. CEDAR: a core-extraction distributed ad hoc routing algorithm. In *Proc. of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 1999*, volume 1, pages 202–209, New York, NY, USA, Mar. 1999.

[107] Ivan Stojmenovic, Mahtab Seddigh, and Jovisa Zunic. Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):14–25, 2002.

[108] Ivan Stojmenovic and Jie Wu. Broadcasting and activity scheduling in ad hoc networks. *Mobile Ad Hoc Networking*, pages 205–229, 2004.

[109] Nguyen B. Truong, Gyu Myoung Lee, and Yacine Ghamri-Doudane. Software defined networking-based vehicular Adhoc Network with Fog Computing. In *Proc. of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1202–1207, Ottawa, ON, Canada, May 2015.

[110] Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. *Wireless networks*, 8(2-3):153–167, 2002.

[111] Deepak Vasisht, Zerina Kapetanovic, Jongho Won, Xinxin Jin, Ranveer Chandra, Sudipta N. Sinha, Ashish Kapoor, Madhusudhan Sudarshan, and Sean Stratman. FarmBeats: An IoT Platform for Data-Driven Agriculture. In *Proc. of the 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017*, Boston, MA, Mar. 2017.

[112] Thiemo Voigt, Martin Bor, Utz Roedig, and Juan Alonso. Mitigating Inter-network Interference in LoRa Networks. In *Proc. of the 2017 International Conference on Embedded Wireless Systems and Networks*, pages 323–328, Feb. 2017.

[113] Mališa Vučinić, Bernard Tourancheau, and Andrzej Duda. Performance comparison of the RPL and LOADng routing protocols in a Home Automation scenario. In *Proc. of the 2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1974–1979, Shanghai, China, Apr. 2013.

[114] Haidong Wang, Brian Crilly, Wei Zhao, Chris Autry, and Sean Swank. Implementing Mobile Ad Hoc Networking (MANET) over Legacy Tactical Radio

Links. In *Proc. of the IEEE Military Communications Conference, MILCOM 2007*, pages 1–7, Orlando, FL, USA, Oct. 2007.

[115] Y.-P. Eric Wang, Xingqin Lin, Ansuman Adhikary, Asbjörn Grovlen, Yutao Sui, Yufei Blankenship, Johan Bergman, and Hazhir S. Razaghi. A Primer on 3GPP Narrowband Internet of Things. *IEEE Communications Magazine*, 55(3):117–123, Mar. 2017.

[116] Brad Williams and Tracy Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *Proc. of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 194–205, Lausanne, Switzerland, Jun. 2002.

[117] Tim Winter et al. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. https://tools.ietf.org/html/rfc6550, Mar. 2012. RFC 6550.

[118] Jie Wu and Hailan Li. A dominating-set-based routing scheme in ad hoc wireless networks. *Telecommunication Systems*, 18(1-3):13–36, 2001.

[119] Li Da Xu, Wu He, and Shancang Li. Internet of Things in Industries: A Survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, Nov. 2014.

[120] Lin Yao, Luning Wang, Lv Pan, and Kai Yao. Link Prediction Based on Common-Neighbors for Dynamic Social Network. *Procedia Computer Science*, 83:82–89, May 2016.

[121] Fan Ye, Gary Zhong, Songwu Lu, and Lixia Zhang. GRAdient Broadcast: A Robust Data Delivery Protocol for Large Scale Sensor Networks. *Wireless Networks*, 11(3):285–298, May 2005.

[122] Hans C. Yu, Giorgio Quer, and Ramesh R. Rao. Wireless SDN mobile ad hoc network: From theory to practice. In *Proc. of the 2017 IEEE International Conference on Communications, ICC 2017*, pages 1–7, Paris, France, May 2017.

[123] Andrea Zanella, Nicola Bui, Angelo Paolo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of Things for Smart Cities. *IEEE Internet of Things Journal*, 1:22–32, 2014.

[124] Muhan Zhang and Yixin Chen. Link Prediction Based on Graph Neural Networks. In *Proc. of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 5171–5181, Montreal, Canada, Dec. 2018.

[125] Ming Zhao, Ivan Wang-Hei Ho, and Peter Han Joo Chong. An Energy-Efficient Region-Based RPL Routing Protocol for Low-Power and Lossy Networks. *Internet of Things Journal*, 3(6):1319–1333, Dec. 2016.

[126] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630, Oct. 2009.

[127] Ming Zhu, Jiannong Cao, Deming Pang, Zongjian He, and Ming Xu. SDN-Based Routing for Efficient Message Propagation in VANET. In Kuai Xu and Haojin Zhu, editors, *WASA*, Lecture Notes in Computer Science, pages 788–797, 2015.

Appendix

SD-MANET CONTROL MESSAGE DESIGN

In this Appendix, we describe the design of the control messages used by all our SD-MANET routing protocols. We have used the RFC 5444 specifications [36] for designing the messages. We first summarize these specifications and then explain the message designs of SD-MANET routing protocols.

A.1 RFC5 444 Specifications

RFC 5444 specifies the syntax for designing messages for exchanging information between the nodes of a mobile ad hoc network. A message structure has four entities: Packet, Message, Address Block, and TLV. We describe them in the following subsections.

A.1.1 Packet

Packet is the top-level entity that includes Packet Header and zero or more Messages. Figure A.1 shows its format.

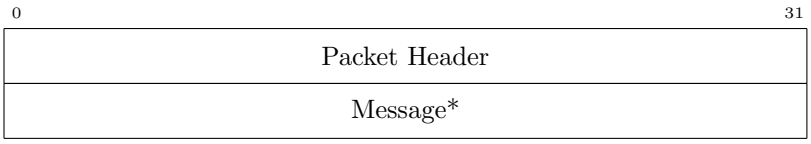


Figure A.1: Packet Format

Figure A.2 shows the format of Packet Header. It is of variable size and includes the following fields:

- **Version:** A 4-bit unsigned integer to specify the design version.

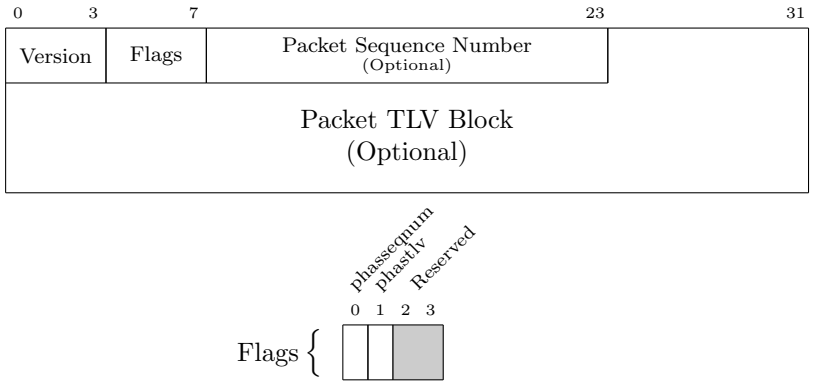


Figure A.2: Packet Header Format

- **Flags:** A 4-bit field to specify the interpretation of the remaining part of Packet Header.
  - **phasseqnum:** If Packet Sequence Number is omitted, then 0, else 1.
  - **phastlv:** If Packet TLV Block is omitted, then 0, else 1.
  - **Reserved:** Flags 2 and 3 are reserved.
- **Packet Sequence Number:** An optional 16-bit unsigned integer.
- **Packet TLV Block:** An optional TLV block. Section A.1.4 describes the structure of TLV Block.

A.1.2 Messages

Message includes Message Header and Message Body. Figure A.3 shows its format.

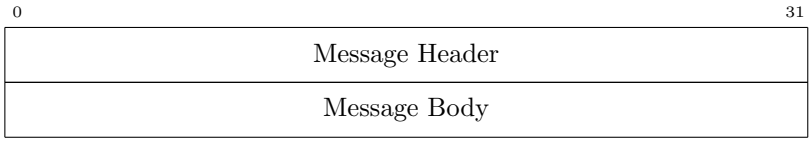


Figure A.3: Message Format

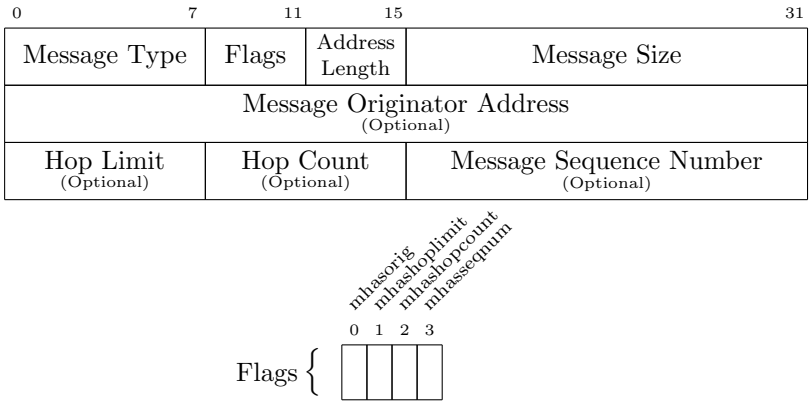


Figure A.4: Message Header Format

A.1.2.1 Message Header

Figure A.4 shows the format of Message Header. It is of variable size and has the following fields:

- **Message Type:** A 8-bit unsigned integer to specify the message type.
- **Flags:** A 4-bit field to specify the interpretation of the remaining part of Message Header.
  - **mhasorig:** If Message Originator Address is omitted, then 0, else 1.
  - **mhashoplimit:** If Hop Limit is omitted, then 0, else 1.
  - **mhashopcount:** If Hop Count is omitted, then 0, else 1.
  - **mhasseqnum:** If Message Sequence Number is omitted, then 0, else 1.
- **Address Length:** A 4-bit unsigned integer to specify the address size. The value is 3 for IPv4 and is 15 for IPv6.
- **Message Size:** A 16-bit unsigned integer to specify the message size, including the size of Message Header.
- **Message Originator Address:** An optional field of size Address Length bytes to specify the message originator.
- **Hop Limit:** An optional 8-bit field to specify the hop limit for the message.
- **Hop Count:** An optional 8-bit field to specify the number of hops traveled.
- **Message Sequence Number:** An optional 16-bit field to specify the message sequence number.

A.1.2.2 Message Body

Message Body is of variable size and includes Message TLV Block and zero or more Address Block and Address Block TLV Block pairs. Figure A.5 shows its format.

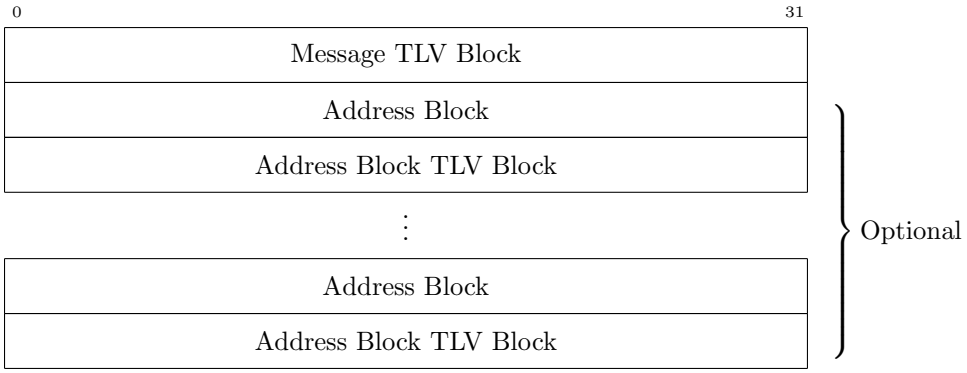


Figure A.5: Message Body Format

Message TLV Block includes the message attribute information in a TLV format. Address Block includes a list of addresses/ address prefixes. Address Block is followed by Address Block TLV Block, which includes attributes specific to the addresses in the corresponding Address Block. Sections A.1.3 and A.1.4 describe Address Block and TLV Block, respectively.

A.1.3 Address Blocks

Address Block includes a list of addresses. The address can be a host address or a subnet address (i.e., address prefix). A field called prefix length determines whether the list contains addresses or address prefixes. The Address Length field in Message Header determines its size.

An address is specified in the Head:Mid:Tail format. There is no semantics associated with Head, Mid, or Tail. The representation allows aggregating addresses with a common prefix. Address Block contains an ordered set of addresses sharing the same Head and the same Tail, but individual Mids. Independently, Head and Tail may

be empty, allowing for representation of addresses that do not have common Heads or Tails.

Address Block can specify a single prefix length for all addresses or an individual prefix length for each address.

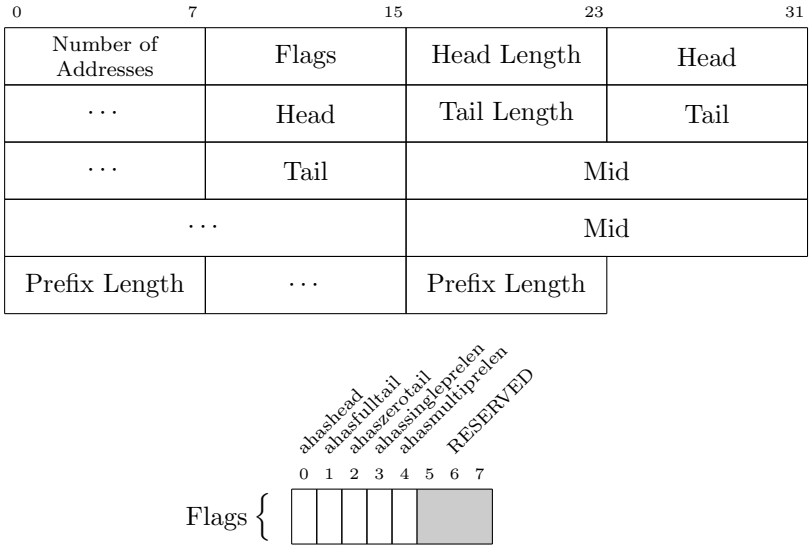


Figure A.6: Address Block Format

Figure A.6 shows the Address Block format. It has the following fields:

- **Number of Addresses:** An 8-bit unsigned integer to specify the number of addresses included in Address Block.
- **Flags:** An 8-bit field to specify the interpretation of the remaining part of Address Block.
  - **ahashead:** If Head Length and Head are omitted, then 0, else 1.
  - **ahasfulltail and ahaszerotail:** Interpretation according to Table A.1
  - **ahassingleprelen and ahasmultiprelen:** Interpretation according to Table A.2
  - **Reserved:** Flags 5-7 are reserved.
- **Head Length:** An optional 8-bit unsigned integer to specify the number of bytes in Head of all addresses in Address Block, i.e., each Head field is Head Length bytes long.

Table A.1: Interpretations of the **ahasfulltail** and **ahaszerotail flags**

ahasfulltail	**ahaszerotail**	**Tail Length**	**Tail**
0	0	not included	not included
1	0	included	included unless Tail Length is zero
0	1	included	not included

Table A.2: Interpretations of the **ahassingleprelen** and **ahasmultiprelen flags**

ahassingleprelen	**ahasmultiprelen**	**Number of Prefix Length fields**	**Prefix length of the n^{th} address prefix in bits**
0	0	0	8 * Address Length
1	0	1	Prefix Length
0	1	Number of Addresses	n^{th} Prefix Length

- **Head:**An optional field, omitted if Head Length is 0; otherwise, it has Head Length leftmost bytes common to all addresses in Address Block.
- **Tail Length:** An optional 8-bit unsigned integer to specify the number of bytes in Tail of all addresses in Address Block, i.e., each Tail field is Tail Length bytes long.
- **Tail:** An optional field, omitted if Tail Length is 0, or if the **ahaszerotail** flag is 1; otherwise, it is a field of Tail Length rightmost bytes common to all addresses in Address Block. If the **ahaszerotail** flag is 1, then Tail Length rightmost bytes of all addresses in Address Block is 0.
- **Mid:** An optional field, omitted if Mid Length (Address Length - Head Length - Tail Length) is 0; otherwise, each Mid is of Mid Length bytes, representing Mid of the corresponding address in Address Block. When not omitted, Address Block contains exactly Number of Addresses $\times$ Mid Length fields.
- **Prefix Length:** An optional 8-bit unsigned integer to specify the length in bits of an address prefix. If the **ahassingleprelen** flag is 1, then a single Prefix Length field is included that contains the prefix length of all addresses in Address Block. If the **ahasmultiprelen** flag is 1, then Number of Address $\times$ Prefix Length fields are included, each of which contains the prefix length of the corresponding

address prefix in Address Block. If Prefix Length value is not present, then each address can be considered to have a prefix length of 8 × Address Length bits.

A.1.4 TLV and TLV Block

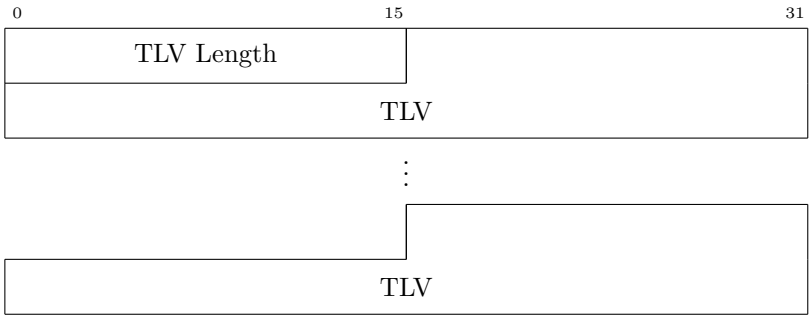


Figure A.7: TLV Block Format

TLV allows associating an arbitrary attribute to Packet, or Message, or with a single address or a contiguous set of addresses in Address Block. Several TLVs can be grouped in a TLV Blocks, with all TLVs within TLV Block associating attributes with either Packet (for TLV Block in Packet Header), Message (for TLV Block immediately following Message Header), or to addresses in the immediately preceding Address Block. Individual TLVs in TLV Block immediately following Address Block associate attributes to either a single address, a range of addresses, or all addresses in Address Block. When associating an attribute to more than one address, TLV includes one value for all addresses or one per address.

Figure A.7 shows the TLV Block format, in which TLV Length is a 16-bit unsigned integer, specifying the size in bytes of TLV Block, excluding the size of the TLV Length field.

Figure A.8 shows the TLV format and has the following fields:

- **Type:** An 8-bit unsigned integer to specify the TLV type.
- **Flags:** An 8-bit field to specify the interpretation of the remaining part of TLV.
  - **thastypeext:** If Type Extension is omitted, then 0, else 1.

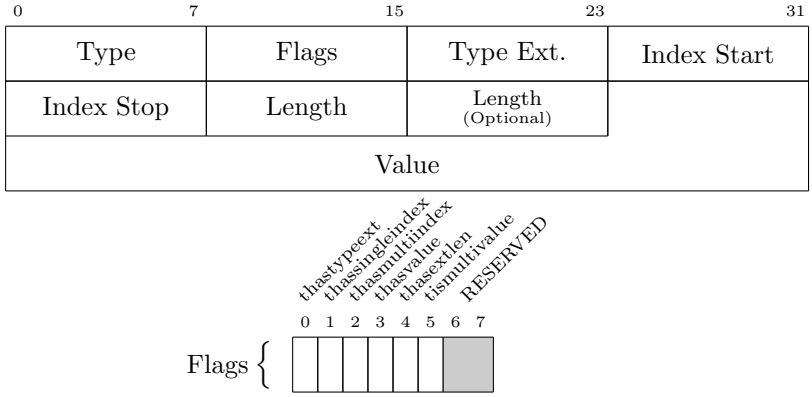


Figure A.8: TLV Format

Table A.3: Interpretations of the **thassingleindex** and **thasmultiindex flags**

thassingleindex	**thasmultiindex**	**Index Start**	**Index Stop**
0	0	not included	not included
1	0	included	not included
0	1	included	included

Table A.4: Interpretations of the **thasvalue** and **thasextlen flags**

thasvalue	**thasextlen**	**Length**	**Value**
0	0	not included	not included
1	0	8 bits	included unless Length is zero
0	1	16 bits	included unless length is zero

– **thassingleindex and thasmultiindex:** Interpretation according to Table A.3.

– **thasvalue and thasextlen:** Interpretation according to Table A.4.

– **tismultivalue:** If TLV includes only a single value, then 1, else 1. This flag is 0 for Packet TLVs and Message TLVs.

– **Reserved:** Flags 6 and 7 are reserved.

Table A.5: Interpretations of the **thassingleindex** and **thasmultiindex flags**

thassingleindex	thasmultiindex	Index Start	Index Stop
0	0	0	For Address Block TLVs, Number of Addresses - 1, otherwise 0
1	0	Index Start	Index Start
0	1	Index Start	Index Stop

- **Type Extension:** An optional 8-bit unsigned integer to specify the TLV TYPE extensions.
- **Index Start and Index Stop:** An optional 8-bit unsigned integer to specify indexes in Address Block TLV. The values interpreted according to Table A.5.
- **Length:** An 8-bit or 16-bit unsigned integer to specify the length in bytes of the Value field in the TLV format.

  Some variables used in the calculation of the Value field are:

  1. *Number of Values:* According to Table A.5, this variable is determined as Index Stop - Index Start + 1.
  2. *Single Length:* If the **tismultivalue** flag is 0, then this variable is determined by Length, else by Length ÷ Number of Values.
- **Value:** In Address Block TLV, the Value field is associated with the addresses from positions Index Start to Index Stop, inclusive. If the **tismultivalue** flag is 0, then the entire field is associated with all addresses. If the **tismultivalue** flag is 1, then this field is divided equally into *Number of Values*, each of length *Single Length* bytes, and associated in order with the addresses.

A.2 SD-MANET Control Messages

We now show the designs of the SD-MANET control messages based on the specifications described in Section A.1. We have used six different control messages in our SD-MANET routing protocols. Table A.6 lists these messages and the protocols using them.

Figure A.9 shows the common Packet Header used in all control messages. Our routing protocols have no requirements for Packet Sequence Number and Packet TLV

Block, so we do not include fields for them in the Packet Header and clear the corresponding flags: **phasseqnum** and **phastlv**.

Table A.6: Mapping between the SD-MANET control messages and the protocols

Control Message	PCC	CORR	CPR	HCPR
Topology Discovery (TD)	Y	Y	Y	Y
Neighbor Information (NI)	Y	Y	Y	Y
Route Request (RR)		Y		
Route Update (RU)	Y	Y	Y	Y
Route Update Acknowledgment (RUA)	Y			
Cluster Information (CI)				Y

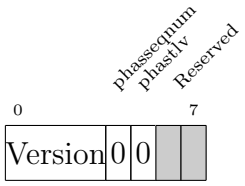


Figure A.9: Packet Header

Figure A.10 shows a generic message with all possible fields in the Message Header and the Message Body. We use this generic message for designing all six control messages.

A.2.1 Topology Discovery (TD)

The SDNC floods the TD message in the network, so there is no requirement for the fields for Originator Address and Hop Count. Figure A.11 shows the TD Message Header.

The TD message includes the previous sender information in the Message TLV Block (shown in Figure A.12). The TD has no Address Block and Address Block TLV Block pairs in the Message Body.

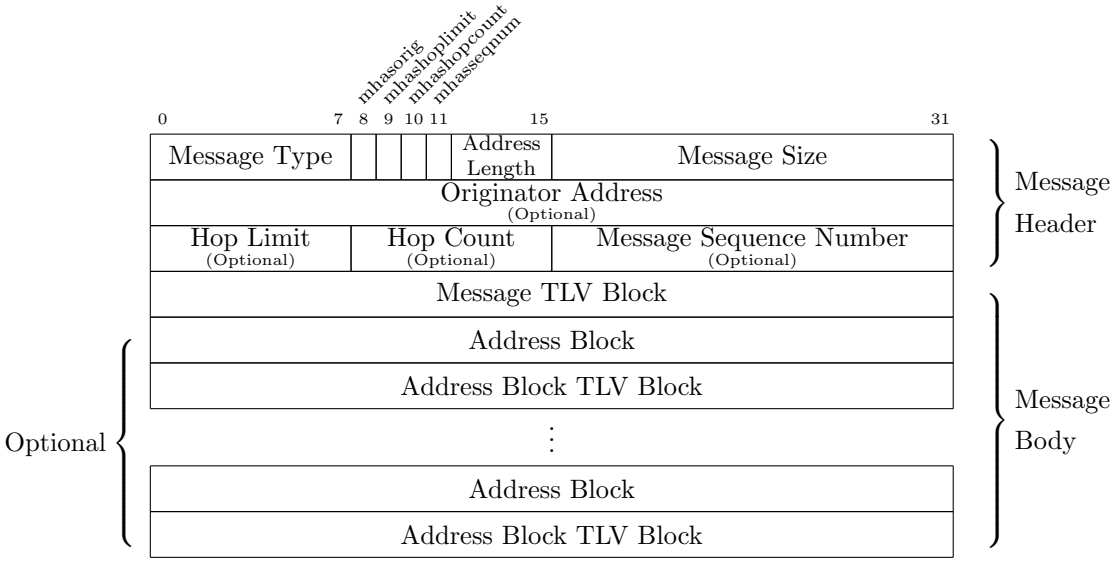


Figure A.10: Message Format

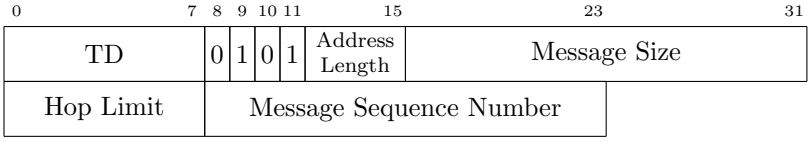


Figure A.11: TD Message Header

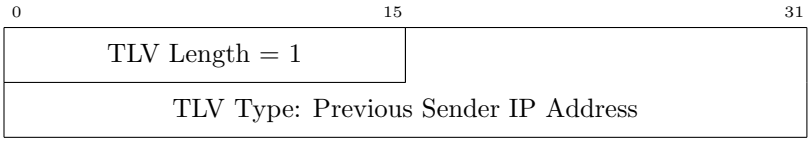


Figure A.12: TD Message TLV Block

We note that all protocols do not use all the fields of the TD message. Table A.7 shows the fields used in each of the routing protocols. All protocols need a field for Message Sequence Number for identifying duplicate TD messages. The CORR, CPR, and HCPR protocols select critical nodes, so they need a field for Previous Sender. The HCPR protocol builds clusters using the Hop Limit field.

Table A.7: Topology Discovery (TD) Message Fields

TD Fields	PCC	CORR	CPR	HCPR
Message Sequence Number	Y	Y	Y	Y
Previous Sender		Y	Y	Y
Hop Limit				Y

A.2.2 Neighbor Information (NI)

All nodes use the NI messages for sending their neighbor information, so is no need for including the fields for Hop Count and Hop Limit. Figure A.13 shows the NI Message Header. The NI Message TLV Block includes the information for the next hop, as shown in Figure A.14.

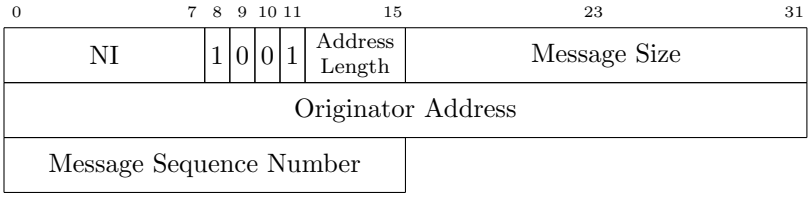


Figure A.13: NI Message Header

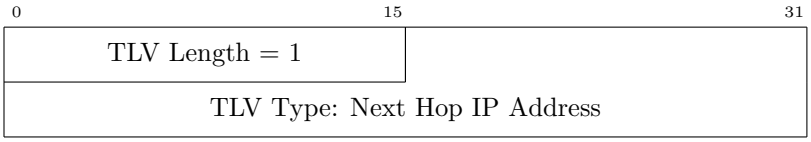


Figure A.14: NI Message TLV Block

The node includes the IP addresses of its neighbor nodes in the Address Block, as shown in Figures A.15.

Figure A.15: NI Address Block

A.2.3 Route Request (RR)

Only the CORR protocol uses the RR message for requesting routes from the SDNC. Figure A.16 shows the RR Message Header. It does not have fields for Hop Count and Hop Limit. Similar to the NI message, the next hop information is included in the Message TLV Block, as shown in Figure A.17.

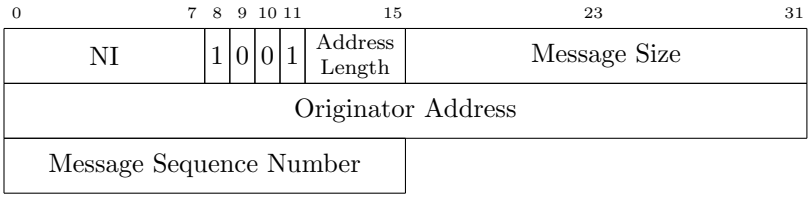


Figure A.16: RR Message Header

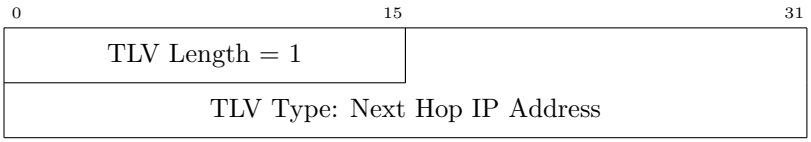


Figure A.17: RR Message TLV Block

A node can request for routes to several destinations. It includes all these destinations in the Address Block, as shown in Figure A.18.

A.2.4 Route Update (RU)

The SDNC uses the RU messages for sending the routing information, so the fields for Originator Address and Hop Count are not needed. Figure A.19 shows the RU Message Header.

Figure A.18: RR Address Block

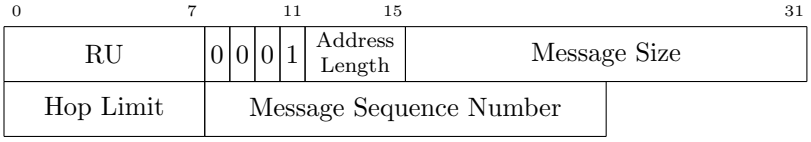


Figure A.19: RU Message Header

The PCC protocol includes the path in the RU message itself. Since the path is a message attribute, the SDNC includes it in the Message TLV Block. Figure A.20 shows the Message TLV Block.

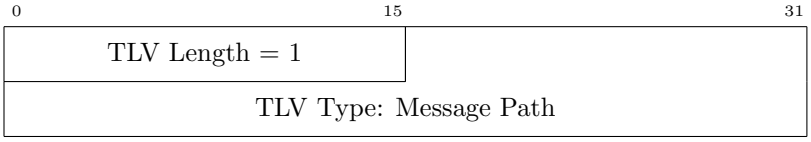


Figure A.20: RU Message Block TLV

The SDNC includes the routing information in the Address Block and Address Block TLV Block pairs.

Consider a scenario in which the SDNC wants to send the routing information shown in Figure A.8. There could be several approaches for encapsulating this information in the Address Block and Address Block TLV Block pairs. One way is to include each destination IP in a separate Address Block and the next hop in its corresponding Address Block TLV Block. So the RU message will have three pairs of Address Block and Address Block TLV Block, as shown in Figure A.21.

Another option is to aggregate the IP addresses having the same next-hop address and include them in a single Address Block and the next-hop IP in the Address Block TLV Block, as shown in Figure A.22.

Table A.8: Routing Information

Destination IP	Next Hop IP
10.0.0.2	10.0.0.9
10.0.0.3	10.0.0.9
10.0.0.4	10.0.0.7

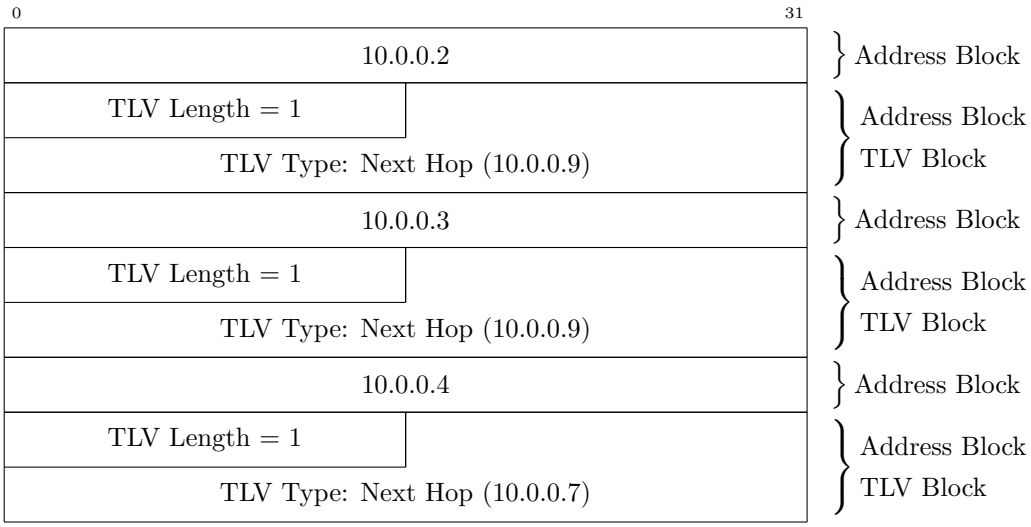


Figure A.21: Each route sent individually.

The third option is to include all destination IP addresses in a single Address Block and all the next-hop IP addresses in its corresponding Address Block TLV Block with a TLV for each next hop, as shown in Figure A.23. The index feature of TLV identifies the mapping between the next-hop IP addresses in the Address Block TLV Block to the destinations in the Address Block.

The SDNC enables flow-based forwarding by sending the routing information in a way that allows nodes to forward data packets based not only on the destination IP address but also fields like IP protocol, TCP/UDP port numbers, and source/destination IP addresses.

Consider a scenario in which the SDNC wants to send the routing information

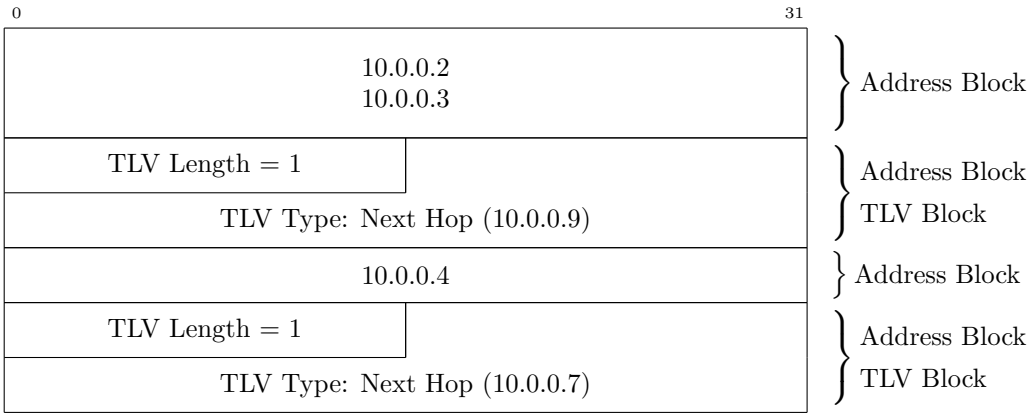


Figure A.22: Destination IP addresses with the same next hop IP address are included in the same address block.

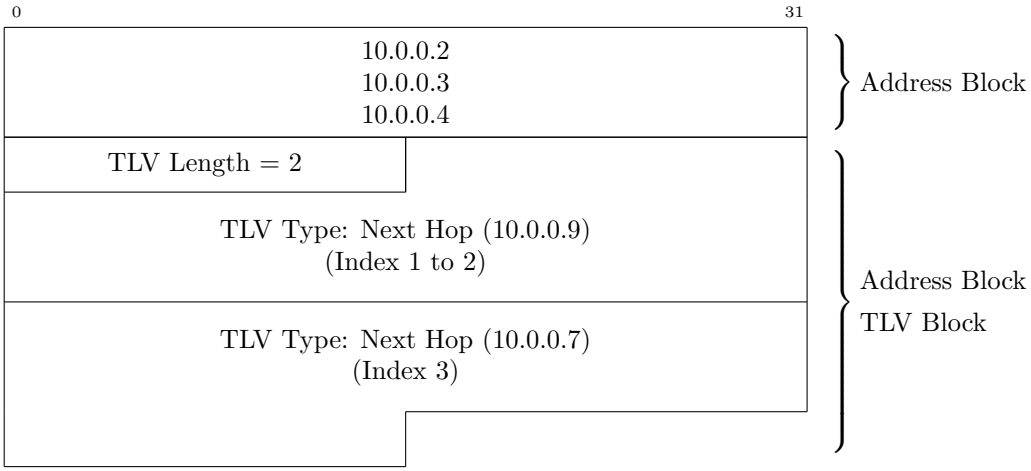


Figure A.23: All destination IP addresses are include in the same address block.

Table A.9: Routing Information

Destination IP	Next Hop IP	IP Proto
10.0.0.2	10.0.0.9	6
10.0.0.2	10.0.0.8	17

shown in Table A.9. The SDNC requires the node to forward the TCP traffic (i.e., IP Proto 6) to the node with IP 10.0.0.9 and the UDP traffic (i.e., IP proto 17) to the

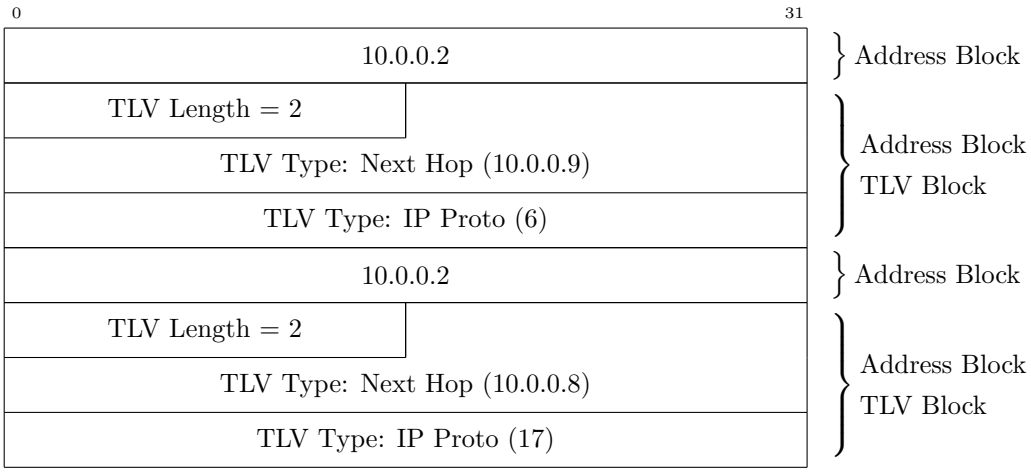


Figure A.24: Routes to forward packets based on multiple fields.

node with IP 10.0.0.8.

The SDNC sends the routing information as shown in Figure A.24. The Address Block TVL Block includes TLVs for both the IP Proto and the next-hop IP address. The node receiving this message installs the rules as shown in Table A.10. It sets the fields which are not in the RU message as wildcards, i.e., do not use these fields to match the packets.

Table A.10: Flow Table

Match Fields		**Actions**
Destination IP	**IP Proto**	
10.0.0.2	6	Forward to 10.0.0.9
10.0.0.2	17	Forward to 10.0.0.8

We note that all protocols do not use all fields of the RU message. Table A.11 shows fields used by each protocol.

Table A.11: Route Update (RU) Message Fields

TD Fields	PCC	CORR	CPR	HCPR
Message Sequence Number	Y	Y	Y	Y
Message Limit				Y
Message Path	Y			
Address Block	Y	Y	Y	Y
Address Block TLV Block	Y	Y	Y	Y

A.2.5 Route Update Acknowledgment (RUA)

The RUA messages acknowledge the received RU messages and do not need fields for Hop Count and Hop Limit. Figure A.25 the RUA Message Header. The next-hop information is included in the RUA Message TLV Block, as shown in Figure A.26. The RUA message does not have Address Block and Address Block TLV Block pairs.

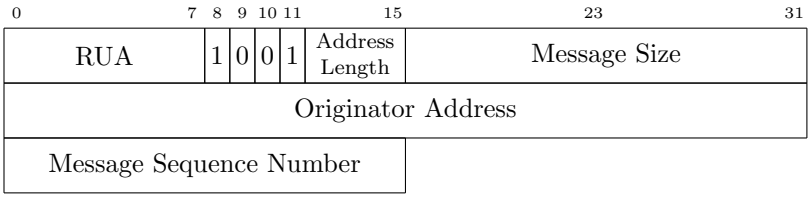


Figure A.25: RUA Message Header

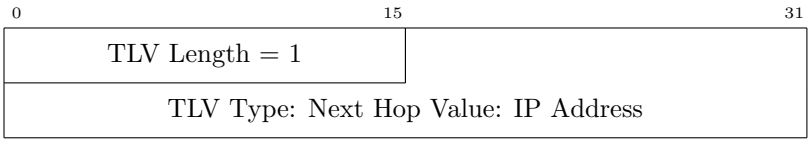


Figure A.26: RUA Message TLV Block

A.2.6 Cluster Information (CI)

In the HCPR protocol, the cluster heads use the CI messages for disseminating their cluster information. The CI Message Header includes fields for Message Sequence

Number and Originator Address, as shown in Figure A.27.

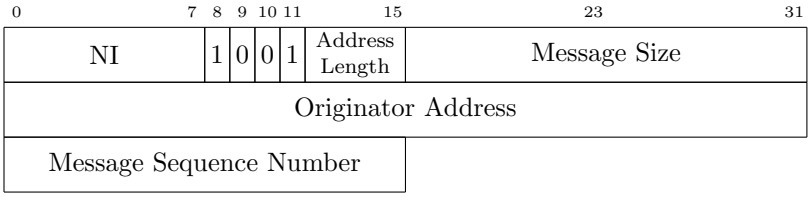


Figure A.27: CI Message Header

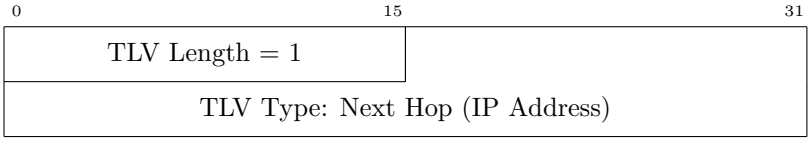


Figure A.28: CI Message TLV Block

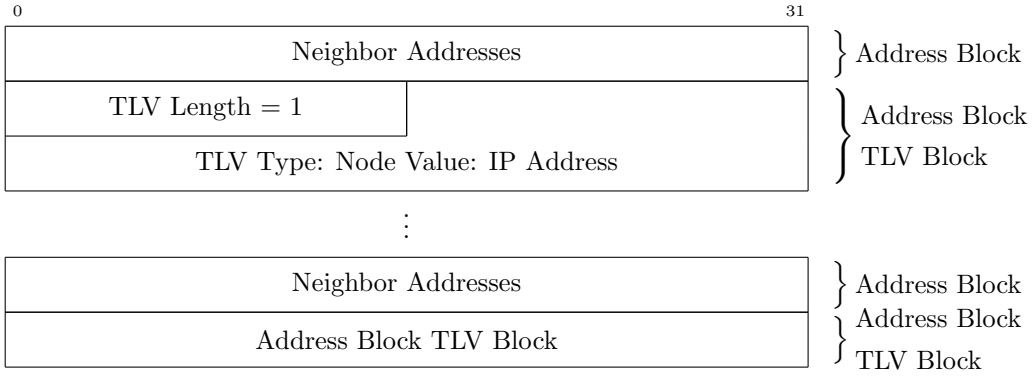


Figure A.29: CI Address Block And Address Block TLV Block Pairs

Similar to the RUA message, the next-hop information is included in the CI Message TLV Block, as shown in Figure A.28. Figure A.29 shows the format in which the cluster information is included in the Address Block and Address Block TLV BLock pairs. The node address is in the TLV Block, and the neighbor addresses are in the Address Block.