NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

AN IN-DEPTH ANALYSIS OF A MACHINE-LEARNING-BASED NETWORK ROUTING PROTOCOL FOR NETWORKING IN A HIGHLY MOBILE TOPOLOGY

by

Jason R. Brown

December 2018

Thesis Advisor:	Justin P. Rohrer
Second Reader:	Geoffrey G. Xie

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE AN IN-DEPTH ANALYSIS OF A MACHINE-LEARNING-BASED NETWORK ROUTING PROTOCOL FOR NETWORKING IN A HIGHLY MOBILE TOPOLOGY	**5. FUNDING NUMBERS**
6. AUTHOR(S) Jason R. Brown	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

This thesis studies the performance of a machine-learning-based DTN routing protocol, QGeo. QGeo is based on the reinforcement learning model called Q-learning whereby an agent in some context takes an action, gains a reward and adapts its decision-making policy based on the reward's value. QGeo is implemented in the ns-3 simulator, and the implementation in this work is based on the previously implemented GAPR protocols. QGeo is then tested in ns-3 alongside GAPR, GAPR2 and GAPR2a, as well as the more commonly known Epidemic, Vector and Centroid DTN protocols. Testing is performed rigorously across four simulation scenarios. The Helsinki scenario simulates mobile traffic in a city, the Omaha and Bold Alligator scenarios simulate amphibious military exercises with various properties, and the Swarm scenario simulates the behavior of a drone swarm based on real-world sensor flight data.

This thesis ultimately shows that QGeo is a highly selective protocol in terms of making forwarding decisions, based primarily in the Q-learning mechanism. This thesis also advances the research previously done at the Naval Postgraduate School in DTN research and development by furthering the testing effort of the protocols that have been implemented. Finally, an added benefit of this study is the incorporation of the Swarm scenario to the DTN testbed, increasing the range of testing capability for comparison of DTN routing protocol characteristics.

14. SUBJECT TERMS disruption tolerant networking, DTN, machine learning, ML, artificial intelligence, AI, routing protocol, QGeo, Epidemic, Centroid, Vector, GAPR, GAPR2	**15. NUMBER OF PAGES** 123
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

AN IN-DEPTH ANALYSIS OF A MACHINE-LEARNING-BASED NETWORK ROUTING PROTOCOL FOR NETWORKING IN A HIGHLY MOBILE TOPOLOGY

Jason R. Brown
Lieutenant, United States Navy
BS, James Madison University, 2011

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2018

Approved by: Justin P. Rohrer
Advisor

Geoffrey G. Xie
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis studies the performance of a machine-learning-based DTN routing protocol, QGeo. QGeo is based on the reinforcement learning model called Q-learning whereby an agent in some context takes an action, gains a reward and adapts its decision-making policy based on the reward's value. QGeo is implemented in the ns-3 simulator, and the implementation in this work is based on the previously implemented GAPR protocols. QGeo is then tested in ns-3 alongside GAPR, GAPR2 and GAPR2a, as well as the more commonly known Epidemic, Vector and Centroid DTN protocols. Testing is performed rigorously across four simulation scenarios. The Helsinki scenario simulates mobile traffic in a city, the Omaha and Bold Alligator scenarios simulate amphibious military exercises with various properties, and the Swarm scenario simulates the behavior of a drone swarm based on real-world sensor flight data.

This thesis ultimately shows that QGeo is a highly selective protocol in terms of making forwarding decisions, based primarily in the Q-learning mechanism. This thesis also advances the research previously done at the Naval Postgraduate School in DTN research and development by furthering the testing effort of the protocols that have been implemented. Finally, an added benefit of this study is the incorporation of the Swarm scenario to the DTN testbed, increasing the range of testing capability for comparison of DTN routing protocol characteristics.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

ARSENL Advanced Robotics System Engineering Laboratory

CAR Context-Aware Routing

CLI Command Line Interface

DLE Drop Least Encountered

DOA Drop Oldest

DoD Department of Defense

DSDV Destination-Sequenced Distance Vector

DTN Disruption/Delay Tolerant Network

e2e end-to-end

ERP-OFDM Extended Rate Physical Orthogonal Frequency Division Multiple Access

FIFO First In First Out

GAPR Geolocation Assisted Predictive Routing

GAPR2 Geolocation Assisted Predictive Routing 2

GAPR2a Geolocation Assisted Predictive Routing 2a

GPS Global Positioning Satellite

GPSR Greedy Perimeter Stateless Routing

GPU Graphics Processing Unit

GUI Graphical User Interface

HMM Hidden Markov Model

IP	Internet Protocol
LCAC	Landing Craft Air Cushion
MANET	Mobile Ad Hoc Network
MASINT	measurement and signature intelligence
MDP	Markov Decision Process
MDR	Message Delivery Ratio
NPS	Naval Postgraduate School
ns2	Network Simulator 2
ns3	Network Simulator 3
PRoPHET	Probability Routing Protocol using History of Encounters and Transitivity
PDR	Packet Delivery Ratio
SARSA	State-action-reward-state-action
SRWBR	short range wide band radio
TCP	Transmission Control Protocol
TTL	Time to live
TTS	Time to send
UAV	unmanned aerial vehicle
UDP	User Datagram Protocol
USG	United States government
USN	U.S. Navy
VANET	Vehicular Ad Hoc Network

Acknowledgments

First, I'd like to publicly thank my Heavenly Father. Without His peace, I would have found myself wallowing in anxiety and unable to move past the fear of being incompetent to complete the heap of work that goes into a thesis. The Apostle Paul's words in Philippians 4:6-7 have never meant as much to me as they did during this time.

Second, there are not words to express the support and care I have felt from my incredible wife, and my wonderful, wild children. Their life-giving joy and sacrificial love for me enabled me to focus on this work when needed, but more importantly, it centered me on the reason I followed the path to pursuing this degree. I'm thankful every day for your love for me, and I'm hopeful that you will be able to see my love reciprocally for you.

Finally, to my advisor, Dr. Rohrer - thank you for the hours spent digging into code, answering questions I didn't even realize I was asking, and clarifying my language in order to effectively communicate my ideas and perceptions. Your communicative competence is astounding, given your technical knowledge. It seems rare to meet someone who is quite so technically gifted, but who can also communicate so effectively. I'm convinced that I learned more through your patient guidance than I would have learned from most any other advisor.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

This chapter discusses the motivation, objectives, scope, limitations and overall structure of this thesis. This thesis attempts to combine three unique fields with major implications for future technological capabilities, both within the context of this study, and the broader networking field overall. This chapter frames the thesis to ensure the context of the study is well understood, while the final chapter presents conclusions and a short discussion of the implications of this study more broadly.

1.1 Motivation

This section explains the motivation for this study through a short discourse on the current state of technology in three unique fields within the broad category of research related to computing technologies. Discussion of robotic swarms, Disruption/Delay Tolerant Network (DTN) routing, and machine learning techniques are each provided.

1.1.1 Robotic Swarms

Drones have become a pervasive technology in modern societies. A quick search on the Internet reveals many uses of commercially available drones, from surveillance to package delivery to hobby sports. The military has begun to test the usability of this technology in various configurations. One of the most promising uses is through swarm deployment of multiple drones. The term swarm generally means that all nodes in the group follow a similar pattern and are controlled centrally; however, it can also be used to encompass multi-robot systems where each node acts independently, cooperating with other robots (nodes) in the system. At the Naval Postgraduate School (NPS) a drone swarm of 50 unmanned aerial vehicle (UAV)s was flown in 2015, with all of the nodes intercommunicating and operating as multiple teams within the 50-node configuration. This has become a testbed for military testing of the usefulness of drone technology, and has promising potential for military operations. Drone swarms rely on efficient communication between all of the nodes in order to interact without collisions, and combine their actions toward a common

goal. This thesis explores useful communication methodologies for this type of interacting system.

1.1.2 DTN Routing

DTN routing is a network routing paradigm described in [1]. Presented as an extension of the Interplanetary Internet, RFC 4838 is meant to describe the architecture of a prototypical DTN. While traditional Internet Protocol (IP) message transport protocols like Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) have been well documented, studied, and employed, they fail to accommodate networks in highly mobile scenarios well. TCP relies on well defined, stable connections, and UDP uses excessive bandwidth. With the advent of high-speed processors, and the use of modern radio technologies, it is apparent that users can connect to a network across a wide variety of mobility scenarios including heterogeneous radios, node movement types and applications. For this reason, it is the belief of this study that DTN research is a field worth pursuing, as it promises a high-upside potential that could prove invaluable not only to commercial users, but in particular the military. In contested or arduous environments where traditional networking schemes are unavailable or not able to be employed, it is very useful to have a network that can accommodate potential disruptions or delays and service many mobile nodes. DTN protocols ultimately represent another step forward in the push for ubiquitous communications availability, which is an invaluable trait for military communications infrastructure, and highly desirable for the everyday user.

While typical routing strategies rely on the connectedness of networks utilizing the TCP/IP framework, DTN routing strategies have significant alternative considerations. In particular, end-to-end (e2e) connectivity is often not immediately available. While a message may have a delivery path from one node across the network to another, that path may be broken at times only to be recovered later. Additionally when a node transmits a message to another node for forwarding to a distant-end destination, the hop-by-hop connectivity may change throughout the lifetime of the message, meaning new routes can open midway through a message's propagation through the network. A message that may have originally been destined to travel through nodes 1, 2, 4, 6 and so on, may end up traveling through nodes 1, 2, 3, 5, 7, and on to its destination. For this reason, DTN strategies tend to make decisions on a hop-by-hop basis, preferring immediate connectivity to e2e paths that may not exist at a later

time in the message's propagation through the network. Because the TCP/IP framework is designed for fully connected networks with dedicated, typically unchanging routing paths, the routing strategies that work on top of that framework are typically unsuitable for DTN routing. Instead, a methodology that takes into account the variable connectivity of the network's nodes is necessary for DTN routing strategies.

1.1.3 Machine Learning

The field of machine learning encompasses a subset of the artificial intelligence community. Machine learning often relies on the concept of a machine learning a policy that is mathematically described to represent the best possible values based on potential decision-making schemes. Often represented by a Markov Model which represents a chain of state-action definitions, the learned policy is implemented in such a way that the machine can autonomously take actions that take the burden off of humans for decision-making. Machine learning can further be split into many subsets to include: supervised vs. unsupervised learning, neural networks, genetic algorithms, and reinforcement learning, to name a few. As it applies to networking paradigms, the focus of this study is to take an in-depth look at reinforcement learning techniques. Reinforcement learning teaches a machine a decision policy by allowing the machine to make a random decision from a distribution of choices, and then administering a reward to shape the bank of possible decisions. This technique can be employed in various ways, but enables the machine to operate without having knowledge of the system in which it is operating prior to execution. Because DTN encompasses such a wide range of independent variables, including various radios, mobility models, etc. it is the belief of this study that reinforcement learning is one of the most promising machine learning techniques for this field.

1.2 Objectives

The main objective of this thesis is to study the application of machine learning techniques to networking strategies, with the intent of discovering whether or not proposed machine learning techniques are a viable option for routing in highly mobile networking environments. The recent growth of interest in machine learning techniques makes it pertinent to study these techniques and their applicability to network science. It is the aim of this study to rigorously test one proposed machine learning routing technique, QGeo, against well

established protocols as a baseline, and against a complex advanced protocol as a measure of its capability.

Since QGeo was initially designed for drone swarms [2], it follows that it should be applicable to any highly mobile network, provided the underlying radio capability exists to support transmission within that network. The initial study tested QGeo's delivery success ratio, overhead, hop count, etc. but the testing was done against other similar protocols, and the scenario in which it was tested was limited. The scenario in the initial paper described testing with a single radio interface, a random mobility scenario, and a single node density parameter [2]. With that in mind, this study tests QGeo as a DTN protocol in the same test environment as other DTN protocols with widely varied parameters and in a larger scope than the initial testing described in [2]. In particular, this study explores QGeo's ability to deliver messages. This encompasses the success rate and delay with which QGeo is able to deliver messages to their destination, QGeo's use of control-level messaging to track topology and connection information, and the associated computational requirements of employing QGeo, to name a few. Further explanation of the specific metrics to be used in testing is encompassed in Section 3.4

1.3 Scope

This thesis tests the effectiveness of a machine-learning-based DTN routing protocol, namely QGeo, against six other protocols across four mobility models, each with unique features. One mobility model is a widely used example of a heterogenous mobile network. Two of the models exhibit military-historical mobility properties with varying node types. The final model exhibits high-speed aerial nodes with high application-level data usage. These scenarios provide a wide-ranging testbed for rigorous testing of new protocols.

In order to effectively accomplish the goals of this thesis, the study only implements QGeo as described in its original presentation [2]. To the keen reader, however, it is an easy transition to incorporate other reinforcement learning schemes into the algorithm, or make slight potential improvements to the reward function. This study did not attempt to model those changes, but leaves them for further study.

1.4 Limitations

One of the main contributions of this study is to implement the QGeo routing protocol in Network Simulator 3 (ns3) from scratch. Since the code from the paper that originally presented QGeo [2] was unavailable, this thesis uses the Geolocation Assisted Predictive Routing (GAPR) protocol implementation as a basis for QGeo. Despite the similarities between them, the paradigms in these two routing protocols are significantly different. GAPR relies on probabilistic inferences, while QGeo bases its decisions on information gained through machine learning based calculations. This study's implementation of QGeo attempts to be consistent with the original presentation; however, it is likely that significant differences exist based on the described differences between GAPR and QGeo. Acknowledgment of the specific differences that are likely to exist is made in Section 3.2.5. All other protocols used in testing were unmodified versions of previously implemented protocols in ns3.

Killeen, Pospischil and Mauldin all acknowledge the difficulty in implementation of the respective mobility models that they employed in their studies, whether it was due to availability of only public data [3], [4] or potential flaws in the utilized real-world data [5]. This study refers the reader to those works for explanation of those difficulties. Based on this study's improvements to the swarm scenario, however, it is noteworthy that a new mobility helper was used in implementing simulator-scheduled mobility events. What that means is that since this study used a new mobility helper to create interpolation of movement points for nodes, the actual node locations are likely to be slightly different from Pospischil's original implementation. However, the data files for the updated swarm mobility helper were built using the same raw data used in [5], so the same challenges exist in that regard. In addition to the mobility helper, this study also employed a new application helper type in the ns3 simulator, which led to challenges in the operation of the DTN protocols. In particular, two of the four applications in the swarm scenario rely on broadcast operation. When the `DTNHelper` is used for applications, an error occurs every time the destination is the broadcast address. For this reason, it is the belief of this study that the DTN implementation in ns3 is lacking some functionality with regard to broadcast; however, it is unknown if that issue lies with the specific routing protocol implementations, or with a more fundamental DTN module in ns3's source code. For scoping purposes, this study was unable to identify the root cause of this issue.

Finally, as described in [3], [4], this study ran at least a thousand simulation runs in testing. Because of the size of overall data produced by all of these runs, it is intractable to analyze each of the output files visually. Therefore, parsing scripts were employed that pulled out the necessary data from the output files and generated plots and charts based on the results. It is necessary to mention, then, that the data described in the analyses of this study is only representative of the raw data, and is not the raw data itself.

1.5 Significant Findings

This study finds that QGeo performs well in scenarios where protocols that excel have the characteristics of long message holding time, and opportunistic forwarding mechanisms. QGeo has a better buffer management scheme than the GAPR protocols because it does not sort the messages in its buffer, but its buffer management suffers compared to Epidemic, Vector, and Centroid because it performs a computation to update the Q-value for every message at every node interaction.

Additionally, this study presents improvements to the Swarm scenario, and ultimately improves the rigor for a growing testbed of simulations for DTN testing. Preliminary results for Swarm improvements were published in [6]. The Helsinki, Omaha, Bold Alligator and Swarm scenarios used for testing all exhibit unique network characteristics to include varying node type, radio parameters and messaging characteristics. This study finds that the growing testbed of simulation scenarios tests a wide range of characteristics and proves to be a rigorous test for all protocols tested.

1.6 Structure of Thesis

Chapter 2 of this thesis discusses the background research related to drone swarm technology and deployment, current DTN routing protocols and strategies, and the current state of Machine Learning technologies with a specific focus on their application to networking technology. The background presentation contains a thorough literature review in each of the respective fields that this study attempts to bring together.

Chapter 3 of this thesis discusses the implementation method for the experimentation setup of this study. In particular, a background and reasoning for the choice of network simulator is explained, and an explanation of previous implementations of protocols and mobility

scenarios is given. After the explanation of the previously implemented routing protocols and mobility scenarios is made clear, a detailed explanation of the implementation method that this study employed is presented.

Chapter 4 of this thesis presents an analysis of the testing and experimentation conducted by this study. In particular, results of the various routing protocols' performance is discussed across all mobility scenarios employed, as discussed in Chapter 3. Findings are presented with graphs and charts in an attempt to support and/or clarify any conclusions drawn from the analyzed data. Conclusions and future work are presented in Chapter 5.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Background

This chapter provides a high-level background in the fields of mobile networking (with specific regard to drone swarm networks), DTN routing techniques, and machine learning (with specific focus on Q-learning). The end of this chapter focuses on recent research published that combines each of these fields in specific ways in order to improve the routing capability of DTN protocols for drone swarm technology and multi-robot systems.

2.1 Highly Mobile Networks

A mobile network is one where the various nodes throughout the network do not maintain a fixed topology. Generally, mobile networks rely on wireless communication infrastructure and the devices must have some embedded or onboard power source. Because of the changing topology and resource-constrained environment, mobile networks have unique communications infrastructure requirements. For instance in aerial networks depending on the scenario (commercial, military, hobby, etc.), node velocity, mobility type (scheduled, predictable, unpredictable), and link layer setup, various networking paradigms may be needed [7]. While the aforementioned parameters are specific to aerial networks, mobile networks in general are each unique and require different communications infrastructure. The distinction of being *highly* mobile only increases the challenging properties of this class of mobile networks.

Highly mobile networks' routing performance is often negatively impacted due to a lack of reliability in data transmission. A DTN architecture is defined as one that "embraces the concepts of occasionally-connected networks that may suffer from frequent partitions and that may be comprised of more than one divergent set of protocols or protocol families" [1]. In multi-robot systems and robotic swarms, the communications infrastructure exhibits characteristics complicit with this definition of a DTN. As has been illustrated by NPS faculty in [8] and [9] and former NPS students in [5] and [3], drone swarm technology in particular suffers from a lack of scalable, reliable communications infrastructure. Power management, network bandwidth, e2e path availability and per-node computational capability are among the many factors that influence a swarm's ability to communicate

reliably. Management of the various factors within an autonomous node has the ability to highly impact the communications capability of each node, thus motivating the need for a low-impact communications infrastructure. In particular, as the size of the drone swarm scales up in node count, the need to reliably communicate efficiently becomes paramount, as discussed in [5].

In observing the operation of communications within a highly mobile disruption-prone network, the characteristics often exhibiting the most influence over the observed poor networking performance include packet latency between nodes (power management and bandwidth availability considerations), short contact times (lack of e2e availability), and high transmission overhead (computation and bandwidth limitations). These issues are common to many MANET and DTN scenarios, and it is noteworthy to mention that in the context of a drone swarm network, the density of aerial nodes determines network type that is employed (DTN or MANET) [7]. Various methods have been introduced to improve network performance in similar environments, to include predictive handover via Hidden Markov Model (HMM) learning in Vehicular Ad Hoc Network (VANET) [10], Ad-Hoc (Mobile Ad Hoc Network (MANET)) protocol development in aerial swarm environments [8], and the application of machine learning (reinforcement learning) to protocol development for mobile robotic networks [2] and wireless sensor networks [11]. In the context of this study, the employment of a routing protocol in a tactical drone swarm network for military application requires the same, if not greater, capability for scalable, reliable communications.

2.2 DTN Routing Protocols

Recent progress in DTN routing protocols has been made through various methods. This section discusses some of those routing protocols as they apply to the testing done in this thesis. In particular, each of the routing protocols discussed highlights the characteristics of a different class, or family, of DTN routing methodologies.

2.2.1 Epidemic

Epidemic is a flooding or “replica based” [12] DTN routing protocol. Proposed by Vahdat and Becker [13] with the goals of “i) maximize message delivery rate, ii) minimize message

latency, and iii) minimize the total resources consumed in message delivery" [13], Epidemic is a wireless protocol that floods messages throughout the network in order to provide e2e connectivity between nodes. As described in *Epidemic Routing for Partially-Connected Ad Hoc Networks* [13], the algorithm utilizes "transitive distribution of messages" between nodes to accomplish its goals i) and ii). When a node generates a message for delivery to another node in the network, it adds that message to its local buffer, which contains a summary vector of every message it has generated or is holding for further delivery (generated by another node). A key point that differentiates Epidemic flooding from a simple broadcast is that this message buffer is constrained in order to limit message delivery latency and resource requirements. In the buffer, each message is identified by a global identifier that is hashed into the summary vector table. In addition to the message buffer, each node maintains a list of hosts with which it has recently been in contact. When two nodes come into contact, they exchange summary vectors in order to decide which messages they have already seen, and which messages need to be exchanged. Once summary vectors are exchanged, each node requests the messages it has not yet seen from the other node. Once summary vectors are exchanged, each node can decide whether it wants to receive the messages it has requested based on various factors such as available local buffer space and incoming message size, etc.

The key points related to Epidemic are that each node maintains a buffer of the messages that it is waiting to flood to other nodes, and when two nodes meet they exchange summary vectors (summary of messages being held) and then the actual messages that each node does not already have. Vahdat and Becker propose First In First Out (FIFO) queuing as the methodology for buffer management in Epidemic [13]; however, they acknowledge that any queue management framework would suffice, provided that the management of messages is done with the summary vector table and correlated buffer. Because this is the case, buffer size is a very important factor in Epidemic. Ultimately, it is fairly straightforward to see that as node count increases, buffer size necessarily increases as well which directly implies a heavy network load for large networks. In *Delay Tolerant Networks*, many methodologies for limiting resource consumption are discussed, to include adding Time to live (TTL) or Time to send (TTS) thresholds, Drop Oldest (DOA) and Drop Least Encountered (DLE) buffer control mechanisms, or an "immunity" list to prevent duplication of store-and-forward behavior for in-contact nodes who have the same buffered messages [14]. Each

of these methodologies addresses a different characteristic of Epidemic's performance, but Epidemic ultimately relies on the assumption of unlimited resources which are not available in practical network systems, especially those characterized as DTN.

2.2.2 Vector

Vector is a "geographic based" DTN routing protocol [14]. Proposed by Kang and Kim [15], Vector employed the use of a node's location and movement history to determine its movement vector. In particular, a node receives its location every Δt seconds and calculates its motion vector for comparison with neighbor nodes' vectors. A determination of the packets that need to be replicated is made based on the two nodes' direction and velocity. Equation 2.1 describes generally how a node determines the number of packets that need to be shared with another node.

$$\text{nFwd}(x) = \gamma f(|\Theta_x - \Theta_y|) + (1 - \gamma)g(|d_x - d_y|)\text{nData}(x) \tag{2.1}$$

In Equation 2.1, nFwd(x) is the number of packets to be replicated, nData(x) is the number of packets that node x is already holding, Θ_x and Θ_y are the directions of movement of nodes x and y, d_x and d_y are the velocities of nodes x and y, function $f(\Theta)$ is the normalized function of Θ and function $g(d)$ is the normalized function of d [15].

Because Vector calculates the number of packets that need to be shared between nodes based on the movement of each node in an interaction, Kang and Kim demonstrated that Vector has a "similar delivery success ratio compared with Epidemic routing, but with 28% less traffic than random way-point model and 38% less traffic than Manhattan mobility model" [14]. In particular, nodes that travel in orthogonal directions transfer the most messages due to their likelihood of contacting different nodes. Additionally, when two nodes are traveling in the same direction at different speeds, packets can be transferred to the faster node. Ultimately Vector provides improvement in node resource consumption through the use of movement information without losing significant performance (compared with Epidemic) in terms of message delivery.

2.2.3 Centroid

Centroid is another geolocation-based routing protocol that adds an improvement in the calculation of a node's location. Proposed by Rohrer [16], Centroid calculates the "center of mass" of a node's position history, rather than trusting the accuracy of the instantaneous GPS-reported location of that node. Rohrer's implementation relies on a time-based calculation that effectively averages out the GPS error over the lifetime of travel of a particular node. Equations 2.2 and 2.3 describe the calculation.

$$C_x(t_p) = \sum_{t=1}^{t_p} \frac{C_x(t-1) \times (t-1)}{t} + \frac{x_t}{t} \tag{2.2}$$

$$\Delta C_x(t_p) = \frac{x_{t_p} - C_x(t_p - 1)}{t_p} \tag{2.3}$$

In Equations 2.2 and 2.3, C_x is the centroid, ΔC_x is the change in centroid since the last time increment, x is the one-dimensional (x, y or z) component of node position, and t_p is the current time increment. ΔC_x is the primary tool used to account for noise factored into historical location data due to GPS errors. [16]. Centroid's performance is comparable to that of Vector, as it is based on the same concepts. Just like Vector, Centroid uses geographic movement information to determine message sharing requirements and limit replicated messages, however it does address Vector's sensitivity to GPS errors. [4].

2.2.4 PRoPHET

Probability Routing Protocol using History of Encounters and Transitivity (PRoPHET) is a probability-based routing protocol. Proposed by Lindgren, Doria and Schelen as a way to improve the delivery rate over Epidemic, while lowering the overall network overhead [17], PRoPHET determines its best next hop per-node by employing a delivery probability metric, and a transitivity factor. Namely, the delivery probability is a measure of how frequently two nodes interact, which has a constant decay rate until the next interaction. The transitivity factor correlates three nodes such that if nodes 1 and 2 are likely to interact, and nodes 2 and 3 are likely to interact, then there is a strong probability that node 1 can transmit messages whose destination are at or through node 3. Equations 2.4, 2.5 and 2.6 represent the main

factors in determining a node's probability of transmission to a single other node.

$$P_{(a,b)} = P_{(a,b)_{old}} + (1 - P_{(a,b)_{old}}) \times P_{init} \tag{2.4}$$

$$P_{(a,b)} = P_{(a,b)_{old}} \times \gamma^{k} \tag{2.5}$$

$$P_{(a,c)} = P_{(a,c)_{old}} + (1 - P_{(a,c)_{old}}) \times P_{(a,b)} \times P_{(b,c)} \times \beta \tag{2.6}$$

For PRoPHET's probability equations, $P_{(a,n)}$ represents the probability of node a delivering to node n, where n is some valid node in the network. P_{init} is either 0 or 1, γ is a constant between 0 and 1, but not including 1, k is a number representing the elapsed time since the last update to $P_(a, b)$, and β is a constant between 0 and 1. Equation 2.4 describes the probability of delivery between two nodes, as a function of their previous probability of delivery and is updated at every interaction. Equation 2.5 represents the aging factor to measure the frequency of interaction of two nodes. Frequently interacting nodes have a higher probability metric, while nodes that meet less frequently have a lower metric, due to the aging equation. Finally, Equation 2.6 handles the transitivity factor in determining follow-on delivery [17].

While this thesis does not implement PRoPHET, it is noteworthy that as a representative of the class of probability-based routing protocols, PRoPHET outperforms Epidemic in terms of reduced network overhead and decreased network delay. PRoPHET performs similarly to Epidemic in terms of message delivery, but it also increases the complexity of routing decisions by adding historical node encounter metrics to determine probability of encounter and performs transitivity calculations.

2.2.5 CAR

Context-Aware Routing (CAR) is an example of a social-based routing protocol. Proposed by Musolesi and Mascolo [18], CAR attempts to solve DTN routing challenges by predicting future system contexts in order to maximize the efficiency of routing decisions. In their

paper, Musolesi and Mascolo define context to be "the set of attributes that describe the aspects of the system that can be used to drive the process of message delivery" [18]. This definition of context allows the flexibility for CAR to measure a system state as a combination of the items in the context set and their corresponding utility values. Based on these values, CAR predicts an overall delivery probability for a given node that attempts to maximize the future context utilities, and sends the calculated probabilities out to connected nodes for routing table updates.

While this thesis does not implement CAR, its benefits as an example of a social-based routing protocol include low network overhead, specifically due to the single-copy nature of the protocol, meaning messages are only forwarded once (in contrast to a flooding scheme, like in Epidemic) [18]. Additionally, CAR illustrates a routing strategy that is resilient to poorly connected networks as locality of nodes is part of the context set. In particular, this methodology allows CAR to outperform many other protocols when a low buffer size is employed because each message does not need to be held for more than one forwarding transmission. Ultimately, the most efficient

2.2.6 GAPR

GAPR is the first in a family of iterated hybrid protocols developed at NPS, each implementing a new feature derived from other protocols or routing methodologies. Proposed by Rohrer and Killeen [19], the initial implementation of GAPR was based on the fact that many routing protocols' performance varies across a range of mobility scenarios. In particular, each protocol has characteristic behaviors that perform better than others in different scenarios. For this reason, GAPR attempted to incorporate the beneficial characteristics of the various protocols and incorporate them into a single protocol, while also adding the benefit of geolocation information to improve performance.

In particular, GAPR employs the following features from various routing protocols as a basis for desirable performance [19]:

- ACK flooding: acknowledgement messages sent out upon successful delivery use flooding to replicate to all nodes in order for buffer space to be made available at each node holding the original message.
- Encounter probability calculation: each node calculates a delivery probability based

on direct encounters with other nodes. When two nodes meet, they use a modified Dijkstra's algorithm to calculate the transitive probability of delivering a message to its destination.

- Short contact exploitation: nodes forward message to a peer node based on a descending order of probability values. In other words, node 1 transmits messages to node 2 in order of the higher probability of a message being delivered to its destination node.
- Buffer management: nodes only delete messages from their buffer when necessary and messages are deleted in order of lowest probability of being delivered. In this way, messages that are more likely to be delivered are held, while messages that are less likely to be delivered are deleted, when necessary.
- Geographic location: upon an encounter, two nodes exchange their current geographic locations and the timestamp of the interaction. The received location information is then entered and saved in a table of encounters for determining a sense of network topology. In addition to the encounter table, nodes maintain and exchange transitive location tables for the purpose of understanding transitive network connectivity.

The above features are implemented by each node running GAPR in the following order of operations upon encountering another network node [19]:

1. Exchange ACK messages and clear local buffers.
2. Forward messages whose destination is the peer node.
3. Exchange location information and routing tables including probability values
4. Modify probability values based on node movement for nodes who have moved suddenly since the last update.
5. Forward messages for whom the peer has a higher probability value in descending order.

The original implementation and testing of GAPR was completed in the ONE simulator [3], [19], with follow-on testing conducted in ns3 [4]. Simulators are discussed in Chapter 3; however, the original findings showed that on average, GAPR performed with better Packet Delivery Ratio (PDR), lower latency and lower overhead. Follow-on testing in ns3 revealed that GAPR is actually doing a lot of background work to build and maintain its various features which ultimately revealed that the measured network overhead increased from the original findings.

Geolocation Assisted Predictive Routing 2 (GAPR2) was proposed by Killeen as a follow-on to GAPR that employed the flooding mechanism of Vector in order to reduce overhead and maintain PDR. Original results showed that GAPR2 reduced GAPR's overhead by five times, and only dropped in delivery ratio by 10%. Further testing in ns3 revealed that depending on the mobility scenario GAPR2 lost about 3%-10% delivery ratio from GAPR and latency increased by 400s to 3100s [4].

Geolocation Assisted Predictive Routing 2a (GAPR2a) was proposed by Mauldin as an improvement to GAPR2 by employing the Centroid mechanism introduced by Rohrer [16]. By introducing the Centroid mechanism to GAPR2's capability, Mauldin demonstrated consistent performance across a range of scenarios, measuring Message Delivery Ratio (MDR) within 1% of GAPR and latency within 600s of GAPR's performance. The performance of GAPR2a is significant; however, Mauldin acknowledges that the drawbacks of GAPR2a include higher network overhead and increased power consumption when compared to GAPR2. Nonetheless, both GAPR2 and GAPR2a perform better than GAPR in terms of overhead and power consumption [4].

2.3 Machine Learning

Various machine learning techniques exist to support the development of decision behaviors by a machine, and many advances in machine learning have been achieved in recent years due to the computational power now available. Most techniques rely on a recorded dataset, either annotated (supervised learning) or non-annotated (unsupervised learning) and train a computer to recognize instances of desired outcomes, in a general sense. For example, Neural Networks (NN) and Deep Learning methods can be applied to train a machine to recognize images for improvements in computer vision technology [20]. Another promising application of machine learning, in this case two techniques – Naïve Bayes and HMM – is in natural language processing [21] by training a machine to automatically differentiate spam email from desired relevant content. The problem with using these methods (and many other machine learning techniques) in automated network routing decision-making is that the fundamental requirement for training is a prescribed data set. In networking, an optimal network design that could be used as a baseline for training a recognition machine is often unavailable, and when applied to highly mobile networks, the variable geographic topology often makes it impossible to have a predefined dataset useful for training [22].

2.3.1 Reinforcement Learning

Despite the challenge of training a machine without a prescribed dataset, it is widely believed that reinforcement learning has promising applications to network optimization primarily because of its ability to develop a policy for an unknown environment [22]. Reinforcement learning is based on the principle of a reward system that guides a trainee to an optimal decision path. As described in the seminal dissertation exploring formal algorithms for reinforcement learning by C. J. C. H. Watkins [23], training a machine through reinforcement learning could be accomplished much like animal training by applying a specific reward loop. In principle, if the trainee makes a correct decision, a positive reward is gained, while an incorrect decision earns the trainee a negative reward. Extrapolating this idea to a more general sense, making decisions that improve the overall system state is analogous to making the "correct" behavioral decision and garner a positive reward, while decisions that degrade the overall system state are analogous to "incorrect" behavioral decisions and decrease the reward factor. Because of this principle of iterative rewards based on system state improvement, predefined examples of the "right choice" are not necessary, thus freeing the requirement of having a prescribed dataset.

Recent examples of the capability of reinforcement learning include training a computer to master a centuries-old board game called Go, and successfully defeating the world champion human player by combining Deep Neural Networks and reinforcement learning techniques [24]. A different system used reinforcement learning and computer vision techniques to view the pixels on the screen and iterate through thousands of attempts to play an Atari 2600 video game, eventually achieving "a level comparable to that of a professional human games tester across a set of 49 game" [25]. Reinforcement learning has demonstrated promising capabilities for training computer systems to complete sequential and constrained problems.

A subset of the broad field of reinforcement learning, as explained in [26], is described as a temporal difference methodology. For scope, this review focuses on the temporal difference family of reinforcement learning strategies; however, other families of reinforcement learning include dynamic programming as policy iteration and rollout-based Monte Carlo methods [26]. Temporal strategies are based on a value function and seek to update their primary value indicator at each time step. All of the temporal strategies rely on a Markov Decision Process (MDP) that characterizes the system as a set of state, action, reward val-

ues. In these methodologies an action in state s takes some action a and gets a reward r. These values are utilized in each algorithm's reward function to update the reward value, which each algorithm seeks to maximize. Equations 2.7, 2.8, and 2.9 describe the reward functions used by common temporal difference learning algorithms.

$$V'(s) = V(s) + \alpha(R(s,a) - \bar{R} + V(s') - V(s)) \tag{2.7}$$

$$Q'(s,a) = Q(s,a) + \alpha(R(s,a) - \bar{R} + Q(s',a') - Q(s,a)) \tag{2.8}$$

$$Q'(s,a) = Q(s,a) + \alpha(R(s,a) - \bar{R} + \max_{a'} Q(s',a') - Q(s,a)) \tag{2.9}$$

In Equations 2.7 - 2.9, the following variables are used:

- s describes a state
- a describes an action
- $\bar{R}$ is the average reward when starting in state s and taking optimal actions a*
- R(s, a) is the reward function
- α is the learning rate. In some literature, this is described as the decay rate.
- V(s) is the function describing the old temporal difference value estimate.
- V'(s) is the function describing the updated (new) temporal difference value estimate.
- Q(s) is the function describing the old state-action Q-value.
- Q'(s) is the function describing the updated (new) state-action Q-value.

Equation 2.7 describes the value update function for the TD(0)-learning algorithm, a generalized temporal difference learning algorithm [26]. Equation 2.8 is the value update function used in the State-action-reward-state-action (SARSA) algorithm [26]. Finally, Equation 2.9 is an alternative notation [26] for the value update function for the one-step Q-learning algorithm proposed by Watkins [23]. While these commonly used temporal difference based reinforcement learning algorithms have been employed in various implementations for a range of machine learning scenarios, they hold promising implications for application in network routing, due to their ability to handle varying states through frequently updating

value functions. Of note, the major difference between SARSA and Q-learning lies in the max function used in Q-learning. While SARSA and TD(0) update their value function based on a randomly chosen action, Q-learning attempts to choose the action that results in the maximum reward, thus maximizing Its value function.

2.3.2 Q-Routing

Q-routing is a reinforcement learning based network routing algorithm. Proposed by Boyan and Littman [27] as an initial application of Q-learning to network routing tasks, it employed an irregular 6x6 grid network topology and showed promising results for the use of reinforcement learning, generally, in the networking field. However, because it was implemented on a non-changing topology, this particular algorithm has no implications for DTN, but acts as a good reference baseline for further reinforcement learning based network routing algorithms.

2.3.3 QGrid

QGrid is a reinforcement learning based network routing algorithm. Proposed by Li et al. in 2014 [28], it attempted to apply the Q-learning algorithm to VANET routing. Qgrid employs unicast routing strategies and geographic location information in order to constrain its routing strategy and make efficient routing decisions. In QGrid, the Q-learning portion, or training, is done before any network routing is implemented. Outside of any networking scenario implementation, the Q-Learning function is defined as follows:

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(f_R(s_t, a_t) + \gamma \max_{a'} Q(f_S(s_t, a_t), a')) \tag{2.10}$$

- S: a discrete set of environment states s_t, where each state represents a grid-box which is an equal division of the total geographic environment in which the network operates.
- A: a discrete set of possible actions a_t that an agent in the network can take. These are routing decisions between grids.
- $f_R(s_t, a_t)$: The reward function describing the immediate reward an agent receives after taking action a_t in state s_t at time t.

- $f_S(s_t, a_t)$: The transfer function describing the resulting state of an agent when action a_t is taken at time t.

Since Equation 2.10 is learned offline prior to any networking scenario implementation, the Q-value is recorded and known prior to the scenario. The Q-learning function is used offline to generate a Q-value table. When nodes in a networking scenario for which the set of states are applicable (ie, a geographically similar region to the region in which the system was trained) attempt to route a packet, the Q-value table is employed to determine the next best *direction* in which to route the packet in order to transmit it to its required destination in the most effective manner. Li et al. state that an "agent infers the environment from the reward *R*" [28] meaning that the Q-value table is derived from the reward *R* for routing from one grid to each of the possible next grids. For instances when the packet's destination is in the grid represented by the next best direction, the reward *R* in QGrid is set to 100. Otherwise, the discount factor γ determines the relative value of the possible directions for routing, which is based on the number of vehicles in each of the possible grids for routing [28]. Ultimately the learning portion of this algorithm relies on the grid-breakdown of the geographic environment rather than on the topology of the network. Then, when the Q-value table is employed in the networking scenario, it is a simple lookup table that each node must query in order to determine its next best grid for routing, followed by a determination of what is the best node in that grid for routing. Determination of the node in the best next grid is made by what Li et al. call the *Vehicle Selection Strategy* [28]. The *Vehicle Selection Strategy* is accomplished by determining if the node who needs to route a pack has only one neighbor. If so, that node is chosen as the node to which the packet will be forwarded. If otherwise, the Q-value table is queried to determine the best next grid for routing. For all of the nodes in the best next grid a conditional probability is determined that is a calculation of all of the possible vehicles' Q-values for their next-hop route. In this calculation, the highest probability calculated wins the routing decision [28] because the highest probability corresponds to the node that is most likely to propagate the packet to its destination.

QGrid was tested against other geolocation-based and grid-based routing protocols. Testing showed that QGrid performed better than all other protocols against which it was tested, specifically HarpiaGrid and Greedy Perimeter Stateless Routing (GPSR). Regardless of its performance, QGrid is intended for VANET implementation, which is a limited set of

DTN scenarios. VANETs describe valid networks for nodes who follow specific routes (in particular, roads and rail tracks). To generalize to the broader class of DTN routing protocols, a different implementation is necessary.

2.3.4 QGeo

QGeo is a reinforcement learning based network routing algorithm. Proposed by Jung, Yim and Ko [2], QGeo takes link *packet travel speed* into account in order to increase reliability of delivery. QGeo relies on in-scenario training rather than offline training, citing the potential for unstable node distribution as a potential cause for errors in the Q-value calculation when training is performed offline. Additionally, rather than breaking the geography into a grid and assigning each grid a Q-value, QGeo assigns each node a Q-value during an online exploration (training) phase, followed by the exploitation phase after Q-learning convergence [2]. The Q-value update equation is described by Equation 2.10, the same as in QGrid. However, the reward function for QGeo is defined as follows:

$$f_R(s_t, a_t) = \begin{cases} R_{max} & when\ s_{t+1}\ is\ destination \\ -R_{max} & when\ s_t\ is\ local\ maximum \\ \frac{f_{pts}}{R_{pts}} & otherwise \end{cases}$$

In the QGeo reward function, R_{max} is the maximum reward value, R_{pts} is a normalized reward value based on the communication range between two nodes and the probability of delivery, and f_{pts} is defined as follows:

$$f_{pts}(\text{diff}_{i,j}, P) = \frac{\text{diff}_{i,j}}{T_{i,j}(P)} \tag{2.11}$$

$$T_{i,j}(P) = (O + \frac{P}{r}) \times \frac{1}{1 - E_{link}} \times \frac{1}{1 - E_{loc}} \tag{2.12}$$

f_{pts} is the function describing the packet travel speed where $\text{diff}_{i,j}$ is the difference between nodes i and j distances to the packet destination. $T_{i,j}(P)$ is the travel packet time calculated in Equation 2.12, where O is the overhead incurred for channel access, P is the packet size, r is the data rate, and E_{link} describes link state error, while E_{loc} describes geo-location error.

The final factor in the Q-value update equation that is distinct to QGeo is the decay factor γ. QGeo allocates γ dynamically utilizing the neighbor distance between neighbor nodes i and j $E[d_{i,j}]$ as:

$$\gamma = \begin{cases} 0.6 & when\ E[d_{i,j}] < d_{comm}. \\ 0.4 & otherwise \end{cases}$$

Similarly to QGrid, QGeo utilizes the Q-value update equation, shown in Equation 2.10, for training the system and creating a Q-value lookup table which is employed during the exploitation phase (equivalent to QGrid online scenario phase) to make routing decisions. Jung, Yim and Ko report that testing revealed QGeo's performance had lower delay, higher control message overhead, lower total network overhead, lower total retransmissions and higher MDR than QGrid and GPSR. Testing was performed in a drone swarm scenario with 25 nodes in a 250m x 250m region with various networking parameters set as described in [2].

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Implementation and Testing Methodology

This Chapter discusses the implementation of the routing protocols and mobility scenarios used in testing, as well as the method for testing. Specifically, Section 3.1 discusses the reasons behind choosing ns3 for testing, and Section 3.2 discusses the implementation of the routing protocols tested, including parameters used, with more elaboration of QGeo as it is a newly implemented protocol in ns3. Section 3.3 discusses the mobility scenarios implemented, presenting a specific case study on the swarm scenario. Finally, Section 3.4 discusses the metrics evaluated in testing.

3.1 Choice of Simulator

Many network simulators exist for testing various DTN networking characteristics. Ultimately, while the ideal scenario is to be able to test any network feature with a simulator that accurately implements all aspects of a given network, it is often intractable or unnecessary to utilize the computing power that would be required to implement every characteristic of a network in a simulation environment. Without a fully-implemented network, then, a desirable characteristic of a simulator to use for testing is the ability to modify the base implementation of the simulator. Of the publicly available simulators that exhibit the characteristic of being modular, or at least modifiable to suit testing needs, two that have been widely used are the ONE and ns3.

3.1.1 The ONE Characteristics

The ONE is a discrete-event network simulator that is widely used for DTN testing. According to Keranen, Ott and Karkkainen, "At its core, ONE is an agent-based discrete event simulation engine" [29]. This means that the ONE implements network characteristics on a per-node basis, and it uses movement modules, routing modules and message generators to build a simulated network. The ONE abstracts the physical and data layers of the network stack in order to efficiently implement network routing along with accurate node mobility in a lightweight package that is easily used for many scenarios.

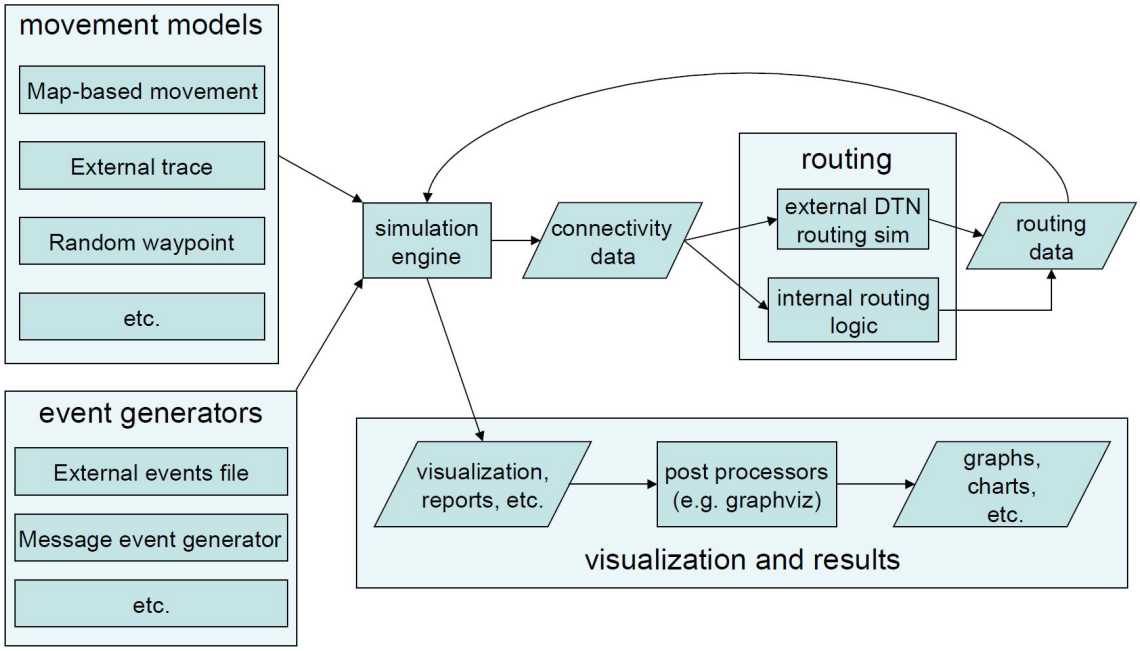


Figure 3.1. The ONE implements DTN routing, using events and movements to drive the simulator engine. Source: [4].

Figure 3.1 illustrates how the ONE simulator uses movements and event generators to drive the mobility engine for the simulator, which feeds the routing module. One of the other helpful features of the ONE simulator is the visual output. The simulator has options to output a mobility map for the nodes' movement in the scenario, as well as network traces at the IP layer, and graphs or charts for easier analysis.

3.1.2 ns3 Characteristics

According to the ns3 manual, "ns3 is a discrete-event network simulator...implemented in C++" [30]. As such, ns3 is implemented as a modular library of C++ modules that can be enabled/disabled, modified or created from scratch. ns3 implements all layers of the network stack through two main modules, called /core and /network [30]. Both of these modules live in the /src directory of the ns3 simulator library. From these modules, many others are implemented to simulate various network implementations, ranging from various routing protocol classes to different network topologies. For the purpose of testing DTN routing protocols, this is a very useful construct because it enables the implementation of

heterogeneous routing, mobility and application models all on top of a common networking core. Figure 3.2 depicts the organization of ns3's software modules.

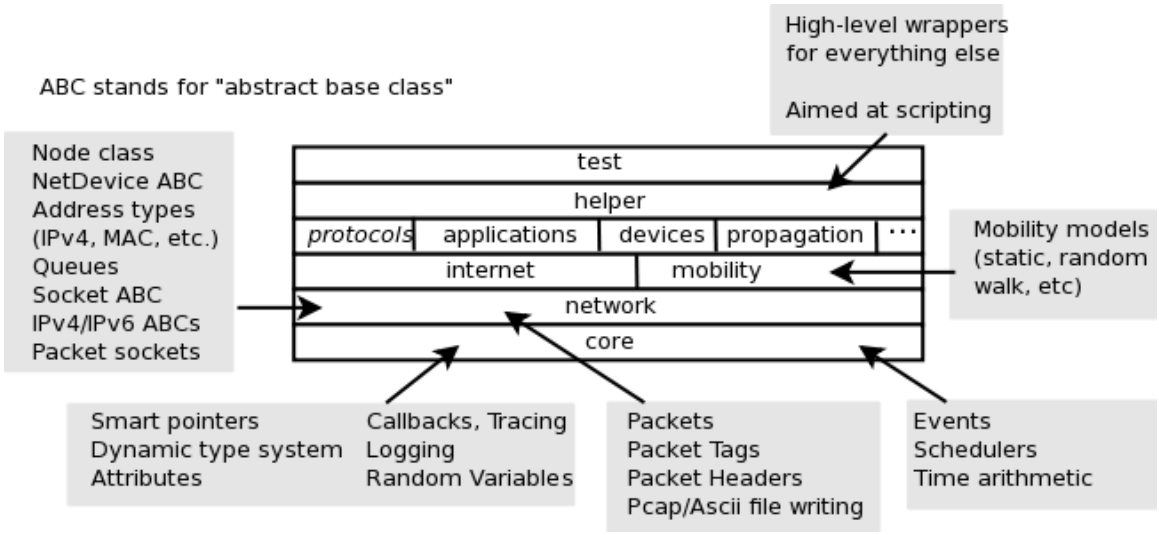


Figure 3.2. The software organization of ns3 implements the entire network stack. Source: [4].

Since ns3 is entirely implemented as a software library, it uses a build tool called Waf to compile all relevant files for a specific simulation, and commence the actual simulation. Waf is comparable to the C language's cmake command. ns3 does not include any Graphical User Interface (GUI) tools so the learning curve to using the simulator is somewhat steep for users unfamiliar with using tools that only have a Command Line Interface (CLI).

3.1.3 Choice of ns3

According to Mauldin's research, ns3's implementation of the entire network stack revealed up to 33% of data transmitted is due to control messaging, which is not implemented in the ONE [4], meaning ns3 is a more realistic network implementation. On the other hand, while ns3's implementation may include more realistic data, it includes no embedded analysis tools, simulations take up to 50 times longer than in the ONE and prototyping new modules is significantly more challenging due to the build environment [4]. Nonetheless, Mauldin's recommendation to test new DTN protocols in ns3 is a useful one as his findings point to the comparison of the realistic nature of the output between the ONE and ns3. Because

ns3 includes the lower layers of the network stack, its output can be interpreted as more realistic; therefore, this study implemented protocol testing in ns3.

3.2 DTN Routing Protocols Implementation

The focus of this study was to test a machine learning based network routing protocol, namely, QGeo. In this study QGeo was implemented in ns3 and tested against Epidemic, Vector, Centroid, and the GAPR family of protocols. This section provides a background to the coding structure of the tested protocols, and an in-depth explanation of the implementation of QGeo.

The ns3 file structure follows common naming conventions. For the DTN protocols, each unique protocol is given a separate folder in the /src directory, identifiable by the name of the protocol. Within each protocol's folder, the following folders are implemented: /doc, /examples, /helper, /model, and /test. Each routing protocol's implementation is contained in the /model folder, and the /helper folder contains files that can be easily used in other models to install the protocol in various simulations. The /doc folder ideally contains an explanation of the protocol including documentation references, /examples contains helpful ns3 tutorial-level examples, and /test contains files for testing the validity of the install for the respective protocol. Within the /model folder, the following naming convention is used for the files corresponding to the DTN protocols that have been implemented ("proto" should be replaced with the name of the protocol being implemented):

- proto-packet.cc(.h): implements the protocol's required packet headers, and serialization/deserialization functions for writing the headers for packet transmission/receipt.
- proto-packet-queue.cc(.h): implements the protocol's queue management scheme. This file also calls proto-packet.cc to access some of the features of the protocol's packet header types. This file implements two classes: `PacketQueue` which is the protocol's buffer, and `QueueEntry` which is an element contained in the buffer.
- proto-routing-protocol.cc(.h): implements the message exchange sequence and required routing functions. This file also calls proto-packet.cc and proto-packet-queue.cc to access various features of the packet headers and queue elements.
- proto-tag.cc(.h): implements simulator-specific metadata tagging utilized by the protocol.

Due to ns3's core implementation, each routing protocol must implement its own version of characteristics that are common to each protocol. In particular, node discovery, acknowledgement messaging, and application-level message handling must be implemented by each protocol. In ns3, these functions were implemented in the same way for each of the tested protocols. Alternatively, each protocol must also implement an exchange sequence, message queueing method, and packet headers corresponding to each protocol's message types. Exchange sequence and headers were implemented in ns3, generally speaking, in unique ways for each of the tested protocols. While there are many commonalities in the queue management scheme of each protocol, there are significant differences in the way that the queue is used, so discussion of each protocol's queue is handled separately. The unique implementations are discussed in each of the protocols' respective descriptions in the following sections.

In the ns3 implementation of the tested DTN protocols, beacon messaging was the method used for node discovery. A node sends out a simple beacon message that is composed of an 8-bit timestamp and 16-bit node identification number. Beacons use node identification number rather than node IP address because a node may have more than one interface, and each interface has an IP address. In order for receiving nodes to know which node sent the beacon, node identification is more useful than one of the node's IP addresses. Beacons are sent by nodes at a regular interval, defined through the scenario-defined parameter `BeaconInterval`. Since a regular beacon interval means nodes transmit at regular times, beacon interval randomness is necessary in order to avoid beacon collisions, since all nodes are transmitting on the same channel. Interval randomness is implemented by the scenario-defined parameter `BeaconRandomness`, which adds an offset to the transmission time of scheduled beacons. All ns3 DTN protocols implement beacons in the same way with the exception of QGeo, which uses a modified beacon header. Details for the QGeo beacon header modifications are explained in Section 3.2.5.

Ack handling in ns3 is implemented by each DTN protocol uniformly. Ack headers are composed of a 64-bit message identification number, 16-bit node identification number, and 16-bit message status field set to Ack. The node identification number in the Ack header is the number of the node that received the message. As mentioned in Mauldin's work, it is noteworthy to recognize that the ns3 DTN Ack is sent each time a node receives a complete message from another node, rather than only sending acks when the destination

receives the messages [4]. This is significant because it is characteristic of the behavior of DTN protocols, where e2e connectivity is often not able to be established; therefore, this Ack behavior can be used to ensure common DTN paradigms such as store-and-forward or flooding are being maintained as nodes receive messages for further transmission when they are not the message's destination. Ack transmission is implemented in the `SendAck` function, which is distinct from the `SendAckSum` function which is used by DTN protocols to reduce overhead by ensuring messages are not retransmitted to a node having already received them.

Finally, the application messaging implementation at the IP layer in ns3 is common among DTN implementations. Each message uses a DTN data packet header, consisting of a 64-bit message identification number, a 16-bit last hop number, a 32-bit total packets number to notify the receiver of how many packets to expect for the current message, and a 32-bit packet index that describes where in the total number of packets expected the current packet fits. For message generation, each node's application generates a UDP message, which is then given a DTN data packet header by the local node before entering the outbound queue. Further details regarding the data packet generation and handling can be found in [4].

3.2.1 Epidemic Implementation

Epidemic was originally implemented in ns3 by Mohammed J. F. Alenazi [31]. Improvements were made by Mauldin in order to improve adherence to the original Epidemic design. Specifically, "the control packet headers, node discovery, and data handling is different" [4].

Epidemic's exchange sequence is handled in the `RecvEpidemic` function and is executed as follows: first, nodes exchange beacons. Upon receipt of a beacon, the two involved nodes determine which one has the lower IP address as means to ensure they don't both send replies at the same time. After deconfliction, the node with the lower IP address sends a `Reply`, including a `SummaryVectorHeader`. Once a node receives a `Reply` it sends a `Reply_back`. Upon receipt of the respective replies, the nodes begin transmitting data messages, based on the `SummaryVectorHeader` exchanged. The `SummaryVectorHeader` contains a list of message identification numbers from the sender's buffer, and it ensures that the sending node does not send data packets that the receiving node already has in its buffer. Nodes send data packets from their message buffer using the `SendDisjointMessages` function,

for which Acks are sent as a reply. Data packets are sent without a control header, and the data packet - ack exchange represents the final state of two nodes' exchange sequence.

As a flooding protocol, Epidemic's queue management scheme is FIFO, meaning messages are entered into the node's buffer in the order they are received, and later those same messages are transmitted in the order that they were received. Queueing of received data packets is handled in the `RouteInput` function. This occurs when a node receives data packets, indicated by the lack of a control header, from another node that has an active connection with a local interface. Packets that are received that contain the Epidemic data packet header are entered into the queue immediately upon receipt.

Epidemic's unique headers are the `TypeHeader`, which defines the packet as an Epidemic packet, and the `SummaryVectorHeader`. Epidemic implements the following headers:

- `Beacon`: Used for node discovery. Composed of an 8-bit message type field set to Beacon, and a 16-bit node identification number.
- `SummaryVectorHeader`: Used to identify `Reply` and `Reply_Back` messages. Composed of a 16-bit fragmentation block, a 16-bit length block to identify how long this message header will be, and a list of 64-bit message identification numbers corresponding to messages that the sending node already has in its buffer.
- `Ack`: Used for message receipt acknowledgment on a per-connection basis. Composed of a 64-bit message identification number, 16-bit node identification number, and 16-bit message status field set to Ack.

3.2.2 Vector Implementation

Vector was initially implemented in the ONE simulator by Kevin Killeen [3], and later transferred to ns3 by Andrew Mauldin [4]. It is also worth noting that the original Vector implementation was built in the ns-2 simulator by Kang and Kim [15]; however, that code is unavailable, so Killeen's implementation was original.

Vector's exchange sequence is handled in the `RecvVector_dtn` function. Beginning with node discovery, when a node receives a beacon, it sends an `AckSum`, which is a list of message identification numbers that have been delivered to their destination. Differing from the `SummaryVectorHeader` in Epidemic, the `AckSum` is a way to more efficiently

reduce overhead incurred by per-connection acknowledgements. Instead of exchanging Acks on a per-connection basis in order to prevent transmission of messages the node already has, nodes send the `AckSum` to tell the other node to remove those messages from its buffer altogether because they have already been delivered. After receiving an `AckSum`, nodes send an `AckSum_Reply`, and any messages in their buffer whose destination is the connected node. The `AckSum_Reply` is the same message format as an `AckSum`. After `AckSum`s are shared and destination messages are sent, nodes exchange `Vector` and `Vector_Reply` messages. These contain the node's movement vector, as well as a list of message identification numbers contained in its buffer. The node receiving a Vector or `Vector_Reply` calculates the number of messages to transmit based on the connected node's movement vector. Once the calculation is complete, a node uses the `SendDisjointMessages` function to transmit messages to the connected node for further forwarding through the network.

In terms of buffer management, since Kang and Kim's paper did not discuss a queueing management scheme [15], Killeen's [3], and later Mauldin's [4], implementation employed a FIFO scheme. As with Epidemic, the `RouteInput` function handles the enqueuing of messages, and the order of message receipt is the only priority given to messages entering and leaving the queue. The `RouteOutput` function handles the dequeuing of messages when they are being sent from a node's buffer, and in Vector the order of sending is based on the oldest-message-first, according to the FIFO management scheme.

Vector's unique headers are the Vector_dtn `TypeHeader` which identifies the message as a Vector message, the `AckSumVectorHeader`, and the `VectorVectorHeader`. Vector implements the following headers:

- `Beacon`: Used for node discovery. Composed of an 8-bit message type field set to Beacon, and a 16-bit node identification number.
- `AckSumVectorHeader`: Used for buffer clearing. Composed of a 16-bit fragmentation block, a 16-bit length block to identify how long this message header will be, and a list of paired 64-bit message identification numbers followed by 32-bit message timestamps corresponding to messages that the sending node knows have already been acknowledged by their destination node (meaning they were delivered to their destination).
- `VectorVectorHeader`: Used for notifying the receiving node of the sender's move-

ment vector and contained messages for forwarding. Composed of a 16-bit fragmentation block, a 16-bit length block to identify how long this message header will be, a 32-bit movement vector x-component block and a 32-bit movement vector y-component block for communicating the sending node's movement vector, and a list of 64-bit message identification numbers corresponding to messages in the sending node's buffer that are destined for further forwarding.

- `Ack`: Used for message receipt acknowledgment on a per-connection basis. Composed of a 64-bit message identification number, 16-bit node identification number, and 16-bit message status field set to Ack.

3.2.3 Centroid Implementation

Centroid was implemented in ns3 by Andrew Mauldin [4]. Based on similarities in operation between Vector and Centroid, Mauldin's ns3 implementation used the ns3 Vector implementation as the baseline for Centroid, making slight modifications based on unique operating characteristics.

Centroid's exchange sequence begins with beacons for node discovery. After a node receives a beacon from another node, nodes exchange `AckSum` and `AckSum_reply` messages. `AckSum`s in Centroid are the same mechanism as in Vector. After a node receives an `AckSum_reply`, it sends a `MsgSum`. `MsgSum` and `MsgSum_reply` messages are the same mechanism as Vectors `Vector` messages; however, instead of using the nodes movement vector, Centroid sends the node's centroid location x and y components. The centroid location is the average traversed location in accordance with the node's movement history. The canonical example of a centroid is that of a node that completes one circular path. The centroid is the center point of the circle that the node made. Alternatively, for a linear movement, the centroid is the middle point on the line between the node's starting location and current location. A detailed explanation of the calculation of a node's centroid can be found in [16]. The receiving node uses the sender's reported centroid to calculate the distance between the two nodes' centroids, and uses that information to determine how many messages to send at the end of the exchange sequence that are purposed for further forwarding. Other than the difference in which location item is sent, Centroid's `MsgSum` and Vector's `Vector` messages are the same. In addition to sending the `MsgSum` or `MsgSum_reply`, nodes also send messages from their buffer whose destination is the connected node at this point in the

exchange sequence. Finally, after transmitting messages destined for the connected node, nodes exchange messages for forwarding purposes that are intended for destinations other than the two connected nodes using the `SendDisjointMessages`.

Centroid's buffer management scheme is the same as Vector's and Epidemic's. Centroid uses FIFO for routing incoming messages in the `RouteInput` function, as well as for routing outgoing messages in the `RouteOutput` function. Because Centroid's FIFO scheme, and message exchange sequence are the same as Vector's, the only significant difference between Vector and Centroid is the message limit calculation performed for the `SendDisjointMessages` function. Vector uses node movement direction, while Centroid uses node centroid location. This is intended to improve reliability of transmission of forwarding messages because the calculation is based on location history, rather than predicted location (based on node vector).

Centroid's unique headers are the Centroid `TypeHeader` which identifies packets to be Centroid messages. The `AckSumHeader` and the `MsgSumHeader` are shared with the GAPR family of protocols. Centroid implements the following headers:

- `Beacon`: Used for node discovery. Composed of an 8-bit message type field set to Beacon, and a 16-bit node identification number.
- `AckSumHeader`: Used for removing messages from a node's buffer. Composed of a 16-bit fragmentation block, a 16-bit length block to identify how long this message header will be, and a list of paired 64-bit message identification numbers followed by 32-bit message timestamps corresponding to messages that the sending node knows have already been acknowledged by their respective destination node.
- `MsgSumHeader`: Used for notifying the receiving node of the sender's centroid location and contained messages for forwarding. Composed of a 16-bit fragmentation block, a 16-bit length block to identify how long this message header will be, a 32-bit centroid x-component block and a 32-bit centroid y-component block for communicating the sending node's centroid location, and a list of 64-bit message identification numbers corresponding to messages in the sending node's buffer that are destined for further forwarding.
- `Ack`: Used for message receipt acknowledgment on a per-connection basis. Composed of a 64-bit message identification number, 16-bit node identification number, and 16-

bit message status field set to Ack.

3.2.4 GAPR Implementation

GAPR and GAPR2 were initially implemented in the ONE simulator by Rohrer and Killeen [3], [19]. Mauldin's work transferred the implementation to the ns3 simulator and also implemented GAPR2a in ns3. Because the GAPR protocols use principles from Vector and Centroid, Vector was used as the basis for the GAPR protocols.

GAPR's exchange sequence begins with the standard beacon node discovery mechanism. After receiving another node's beacon, nodes running one of the GAPR protocols send an `AckSum` message, the same as Vector and Centroid. Following `AckSum` exchanges, nodes send `Probs` and `Probs_reply` messages which contain a list of interaction probabilities. The list of probabilities is a running database that each node keeps locally and updates when another node is encountered. The probabilities are incremented based on frequency of encounter between two nodes and are normalized. In addition to `Probs` and `Probs_reply` messages, nodes send messages whose destination is the connected node at this point in their exchange sequence. After receiving the `Probs` and `Probs_reply` messages, nodes exchange their local transitive probability tables for deconfliction via a `Trans_Probs` or `Trans_Probs_reply` message. `Trans_Probs` and `Trans_Probs_reply` messages include the sending node's table of probabilities that it has received via interaction with other nodes. When a node receives another node's transitive probability table, if the same record exists in both tables, then the record with the most recent timestamp between the local and received tables is chosen. After exchanging `Trans_Probs` and `Trans_Probs_reply` messages, nodes exchange `MsgSum` and `MsgSum_reply` messages, which are the same as the Centroid `MsgSum` and Vector's `Vector` messages. GAPR and GAPR2 send the node vector components in the `MsgSum` headers, while GAPR2a send the node centroid components. After exchanging `MsgSum` messages, nodes finally send messages that are intended for further forwarding, based on varying schemes between the different GAPR protocols, as explained below.

When a node running one of the GAPR protocols receives a `MsgSum`, certain calculations are performed based on the version of GAPR that is running. GAPR nodes perform no calculation with the information, and proceed to send all messages in the buffer that

require further forwarding. GAPR2 nodes use the provided vector components to calculate the message forwarding limit based on predicted movement and angle of incidence, as is done in Vector. GAPR2a nodes use the provided centroid location components to calculate the message forwarding limit based on centroid location as is done in the Centroid protocol. With these mechanisms, it is easy to see a strong parallel in the development of the GAPR2 and GAPR2a protocols with the Vector and Centroid protocols, respectively. Namely, GAPR establishes protocol uniqueness based on the exchange of probability and transitive probability tables, while GAPR2 implements Vector's message limit calculation for forwarding messages with non-local destinations, and GAPR2a implements Centroid's modification to the message limit calculation.

One of the main features that differentiates GAPR from the previously described protocol implementations is its queue management scheme. Instead of FIFO, GAPR protocols all implement a queue prioritization based on hop count and delivery probability. Delivery probability information is calculated based on the exchanged transitive probability tables. The local node's transitive probability tables is managed by a class that is unique to GAPR called `MeetingProbabilitySet`, which is found in the file gapr-meeting-prob.cc(h). Hop count is used to determine the threshold for calculating which messages to prioritize. Message prioritization and queue re-organization is conducted in the `gaprMergeSort` function, which is called in the last step of the exchange sequence, when nodes determine which messages to send for further forwarding (`SendDisjointMessages`).

GAPR's unique headers are the GAPR `TypeHeader` which identifies packets as GAPR messages, the `Probs` header and the `Trans_Probs` header. GAPR protocols implement the following headers:

- `Beacon`: Used for node discovery. Composed of an 8-bit message type field set to Beacon, and a 16-bit node identification number.
- `AckSumHeader`: Used for removing messages from a node's buffer. Composed of a 16-bit fragmentation block, a 16-bit length block to identify how long this message header will be, and a list of paired 64-bit message identification numbers followed by 32-bit message timestamps corresponding to messages that the sending node knows have already been acknowledged by their respective destination node.
- `ProbsHeader`: Used for notifying another node of the sending node's probability of

encounter for each node in its local probability table. Composed of a 32-bit local node's location x coordinate, 32-bit local node's location y coordinate, 32-bit local node's time of last probability update, 16-bit fragmentation block, 16-bit record count block to notify the receiver of the length of this message, and a list of record blocks which are defined as follows. A single record in the `ProbsHeader` contains the sending node's knowledge of another node in the form of a 32-bit node IP address, 32-bit node location x coordinate, 32-bit node location y coordinate, 32-bit node time of contact, and 32-bit node delivery probability.

- `TransProbsHeader`: Used for sharing transitive probability tables with the purpose of deconfliction and updating more recent information. The `TransProbs` stage of the exchange sequence is composed of groups of records corresponding to other nodes' knowledge of other nodes. Each record is transmitted with a separate `TransProbsHeader` composed as follows. 32-bit record node's IP address, 32-bit record node's time of last probability update, a More Records flag to notify the receiver if more records are contained in the sending node's transitive probability table, a Fragmentation flag to notify the receiver if there are more messages corresponding to this record, a 16-bit number of probabilities that notifies the receiver of how to deserialize this record, and a list of 32-bit transitive node IP address and 32-bit delivery probability pairs.
- `MsgSumHeader`: Used for notifying the receiving node of the sender's vector/centroid location and contained messages for forwarding. Composed of a 16-bit fragmentation block, a 16-bit length block to identify how long this message header will be, a 32-bit vector/centroid x-component block and a 32-bit vector/centroid y-component block for communicating the sending node's centroid location, and a list of 64-bit message identification numbers corresponding to messages in the sending node's buffer that are destined for further forwarding.
- `Ack`: Used for message receipt acknowledgment on a per-connection basis. Composed of a 64-bit message identification number, 16-bit node identification number, and 16-bit message status field set to Ack.

3.2.5 QGeo Implementation

The original implementation of QGeo was written by Jung, Yim and Ko [2]; however, none of the code described in their paper is available. Therefore, one of the main efforts

of this study was to translate the description into a viable and consistent implementation. Additionally, the testing described in [2] was minimal, but showed promise, so this study aimed to expand on that testing. Based on the original paper, and previous work in ns3, this study implemented QGeo by using Mauldin's GAPR implementation as a baseline. The similarities between QGeo and GAPR are the predominance of a geolocation factor in the protocol's behavior, as well as a separate metric: Q-value (QGeo) and Meeting Probability (GAPR), in the next-hop calculation, when a node is transmitting messages. However similar these protocols may be, significant differences exist. As the primary example of the difference in the protocols, the operations on the buffer are completely different between GAPR and QGeo. QGeo calculates Q-values based on the buffer's current values (explained below), while GAPR sorts the messages in the buffer based on their probability comparison.

QGeo shortens the exchange sequence from GAPR by removing the `Probs` and `Trans_Probs` messages from the exchange. However, it adds initial overhead by modifying the beacon header. It is important to understand the way that QGeo uses Q-values prior to understanding the entire exchange sequence. Node discovery in QGeo starts with beacons; however, the beacon in QGeo includes the sending nodes' list of local Q-values, which represents the relative likelihood of a node to be able to transmit to each other node in the network. Many times the Q-value of a single node in the local Q-value table is 0. When nodes receive beacons, the transmitted Q-value table is used to update the receiving node's transitive Q-value table, which is managed by the `QValueSet` class, maintained in the file qgeo-qtable.cc(.h). The transitive Q-value table is a node's map of all other nodes and their known Q-values to each other destination. In this way, the `QValueSet` is very similar to GAPR's `MeetingProbabilitySet`. The difference is in the method of implementation, and use of the associated Q-values (QGeo) or probabilities (GAPR). In addition to the Q-value table, the QGeo beacon header also includes location information, link error and location error [2]. To implement link error, this study utilized the `WifiNetDevice` classes `GetFrameErrorRate` method, and to implement location error, this study used a hard value of 20%, which is the max value in the initial paper for QGeo [2]. Because the local Q-value table and other values are included in the beacon header, beacons in QGeo have a higher bit-size, which is dependent on the size of the network. Additionally, it is worth noting that Q-values received via a beacon are directly entered into the receiving node's transitive Q-value table. Local Q-values are updated via the Q-value update equation, described in

Equation 2.3.4, later in the message exchange sequence, when nodes calculate the messages to be sent for forwarding purposes.

With an understanding of how Q-values are used in QGeo, it is easy to understand the protocol's exchange sequence. First node discovery is implemented via the modified beacons. Once a node receives a beacon and updates its transitive Q-value table, it sends an `AckSum` message. Once nodes exchange `AckSum` and `AckSum_Reply` messages, nodes send messages whose destination is the connected node, and `MsgSum` messages. Finally, after exchanging `MsgSum` and `MsgSum_Reply` messages, nodes use the `SendDisjointMessages` function to send messages for further forwarding. This point of the exchange sequence also notes a difference from GAPR's operation, as GAPR uses a queue management scheme that implements a reorganization of the buffer for each connection state and is based on each message's destination at this point. QGeo, on the other hand, does not sort the buffer here, thus making computational requirements at this point in the exchange sequence significantly lower.

As noted above, QGeo does not implement a buffer scheme based on message delivery probability as in GAPR. Instead, QGeo relies on the FIFO management scheme like Epidemic, Vector and Centroid. While QGeo does no sorting on its buffer, it does update the qvalue for each node based on the each message's destination using the Q-value update equation described in Section 2.3.4. One area where this implementation may significantly differ from the initial implementation by Jung, Yim and Ko is that this study summed the `WifiMac` classes `GetSlot` and `GetSifs` methods to define the overhead term. This may not be a completely accurate model of the network access overhead between two nodes, but it is the belief of this study that the performed calculation gives at least a minimum overhead access requirement value that is valid. Due to QGeo's buffer update behavior, computational requirements are higher in QGeo's buffer management scheme than a simple FIFO scheme, but does not require sorting, which allows it to be less complex than that of management schemes that do require sorting. This is an important note in the use of reinforcement learning for networking: while it is more computationally complex than traditional, simpler schemes like FIFO, it attempts to manage that complexity by incorporating a reward function for learning a good selection policy. In the case of QGeo, the reward-based selection policy is implemented in the buffer, and only used when searching the buffer for messages whose best next hop is the node to which the sender is connected.

QGeo's only uniquely named header is the QGeo `TypeHeader`, which identifies packets as QGeo messages; however, QGeo's `Beacon` header is very different than the previously discussed protocols' implementations. QGeo implements the following headers:

- `Beacon`: Used for node discovery, Q-value sharing and reporting of node location, location error, and estimated link error. Composed of an 8-bit message type field set to Beacon, a 16-bit node identification number, a 64-bit node location x coordinate, a 64-bit node location y coordinate, a list of pairs composed of known 32-bit node identification number followed by a 32-bit Q-value, and finally a 32-bit link error estimate and 32-bit location error. **The original QGeo paper removed nodes from the known local list of Q-values if a beacon was not received from that node within two beacon intervals [2]; however, this study did not remove nodes from the known Q-value list. This implementation means that if a message's best Q-value belongs to a node that is out of range, the message is held in the sender's buffer until a better Q-value is found. Additionally, if the best value is shared between two nodes, this study's implementation chooses the node whose Q-value was seen first.
- `AckSumHeader`: Used for removing messages from a node's buffer. Composed of a 16-bit fragmentation block, a 16-bit length block to identify how long this message header will be, and a list of paired 64-bit message identification numbers followed by 32-bit message timestamps corresponding to messages that the sending node knows have already been acknowledged by their respective destination node.
- `MsgSumHeader`: Used for notifying the receiving node of the sender's movement vector and contained messages for forwarding. Composed of a 16-bit fragmentation block, a 16-bit length block to identify how long this message header will be, a 32-bit vector x-component block and a 32-bit vector y-component block for communicating the sending node's centroid location, and a list of 64-bit message identification numbers corresponding to messages in the sending node's buffer that are destined for further forwarding.
- `Ack`: Used for message receipt acknowledgment on a per-connection basis. Composed of a 64-bit message identification number, 16-bit node identification number, and 16-bit message status field set to Ack.

3.3 Mobility Scenarios

A significant contribution of this study was to test DTN protocols in a robotic swarm scenario. Previous swarm implementation by Pospischil [5] was improved and utilized for testing all DTN protocols. This section gives a background to the Helsinki, Omaha, Bold Alligator and Swarm mobility scenarios as well as a short, yet detailed case study for the improvements made to the swarm scenario.

The mobility scenarios described below have various roots. Nonetheless, as map-based models, each scenario utilizes the Network Simulator 2 (ns2) mobility format to generate node mobility. The Helsinki, Omaha and Bold Alligator scenarios' ns2 mobility files were all generated as standard output files from the ONE simulator as described in [4]. Killeen initially implemented the Helsinki and Bold Alligator scenarios in the ONE [3], while Mauldin implemented the Omaha scenario in the ONE [4], and utilized the output files from all three scenarios in the original ns3 implementations of these mobility models. The Swarm scenario, initially implemented by Pospischil, translated real-world drone swarm flight Global Positioning Satellite (GPS) logs into the ns2 mobility files necessary for drone mobility implementation in ns3 [5].

3.3.1 Helsinki

The Helsinki scenario is the canonical mobility model in DTN simulation testing. Based on the city of Helsinki, Finland, the scenario employs three types of nodes that represent real-world urban radios. Pedestrians are the slowest-moving nodes and they move on the map's sidewalks. Cars are the nodes with the moderate speeds and they move on the map's roads. Finally, trams are the fastest moving node types and they only move on the map's tram tracks.

Figure 3.3. The Helsinki mobility scenario base implementation map. Source: [4].

Figure 3.3 is the map used in the ONE simulator to conceptually initiate node movement according to the previously identified node types [29]. Based on the map, nodes are assigned movement commands along the lines of the map which correspond to tram tracks and roads. Pedestrians move along the road lines, at a slower speed than the cars. For testing purposes, ten different mobility scenarios are used with the Helsinki map implementation.

For consistency of testing in order to thoroughly examine the performance of QGeo, testing parameters in the Helsinki scenario are the same as previous testing on DTN protocols in ns3 as conducted in [4]. Table 3.1 is taken from [4] and describes the radio parameters used in the Helsinki scenario.

Table 3.1. Helsinki scenario parameters. Adapted from [4].

Parameter	**Values**
Simulation Duration	12 hrs
Simulator Seed	717, 718
Warmup Time	1000 s
Timestamp Resolution	0.1 s
Beacon Interval	5 s
Number of Pedestrians	80
Number of Cars	40
Number of Trams	6
Base Radio Bandwidth (Mbps)	54
Tram Radio Bandwidth (Mbps)	526.5
Base Radio Transmit Range	10 m
Tram Radio Transmit Range	1000 m
Base Buffer Size (MB)	5
Tram Buffer Size (MB)	50
Message Rate	1 / 25 - 35 s
Message Size	0.5 - 1.0 MB
Message TTL	5 hrs
Hop Limit	50
Protocols	Epidemic, Vector, Centroid, GAPR, GAPR2, GAPR2a, QGeo

3.3.2 Omaha

The Omaha scenario is a custom mobility scenario that implements a model of the World War II D-day invasion of Normandy, at the Omaha beachhead. The Omaha scenario is unique because it presents an historical event as the basis for mobility. While specific movements were not recorded, numbers of troops, vehicles and vessels are known, and the geography of the area is known, so approximate movements are implemented, as described in [4]. The initial implementation by Mauldin was completed in the ONE simulator, with the resulting ns2 mobility file being the basis for the ns3 scenario implementation. The node types in the scenario are infantry troops that move on foot, ships that move over water, and smaller vessels that move along the water and land at the beachhead, which carry the troops. For this reason, the smaller vessels move at the same speed as the troops. Troop ground

movement (after debarking from the vessel) maintains the same speed. In this scenario, all nodes use the same radio and associated parameters.

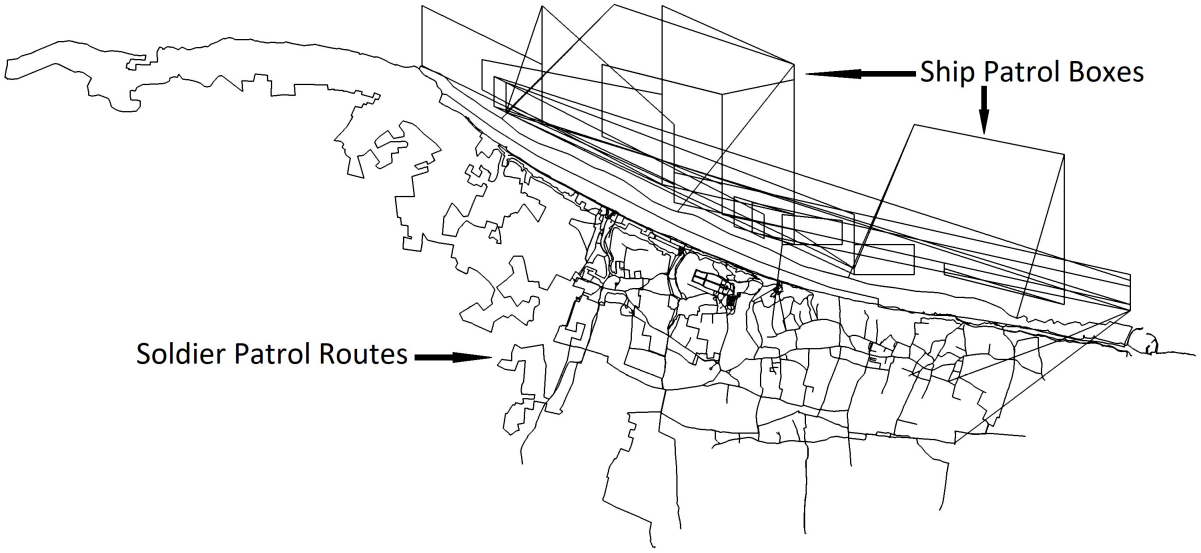


Figure 3.4. The Omaha mobility scenario base implementation map. Source: [4].

Figure 3.4 is the map used in the Omaha scenario implementation. The labeled ship patrol boxes represent areas that ships and smaller vessels traverse over water, while the labeled troop patrol routes are followed by infantry troop nodes over land. In testing, ten different mobility files are used on the same map, with the same number of nodes. Table 3.2 is taken from [4] and describes the radio parameters used in the Omaha scenario.

Table 3.2. Omaha scenario parameters. Adapted from [4].

Parameter	**Values**
Simulation Duration	12 hrs
Simulator Seed	717, 718
Warmup Time	1000 s
Timestamp Resolution	0.1 s
Beacon Interval	5 s
Number of Soldiers	44
Number of Ships	17
Radio Bandwidth	12 Mbps
Radio Transmit Range	550 m
Soldier Node Buffer Size (MB)	5
Ship Node Buffer Size (MB)	50
Message Rate	1 / 25 - 35 s
Message Size	0.5 - 1.0 MB
Message TTL	5 hrs
Protocols	Epidemic, Vector, Centroid, GAPR, GAPR2, GAPR2a, QGeo

3.3.3 Bold Alligator

The Bold Alligator scenario is another custom built, military-based scenario. The Bold Alligator scenario attempts to model the movements of a military-assisted humanitarian extraction operation in a disaster scenario. It does this by using modified plans from a real-world military exercise to generate node movements. The original implementation was built by Killeen in the ONE simulator [3]. His implementation used ships, Landing Craft Air Cushion (LCAC), Humvees, ground troops and helicopters for node types. Mauldin transferred the ONE's ns2 output file as the ns3 base model.

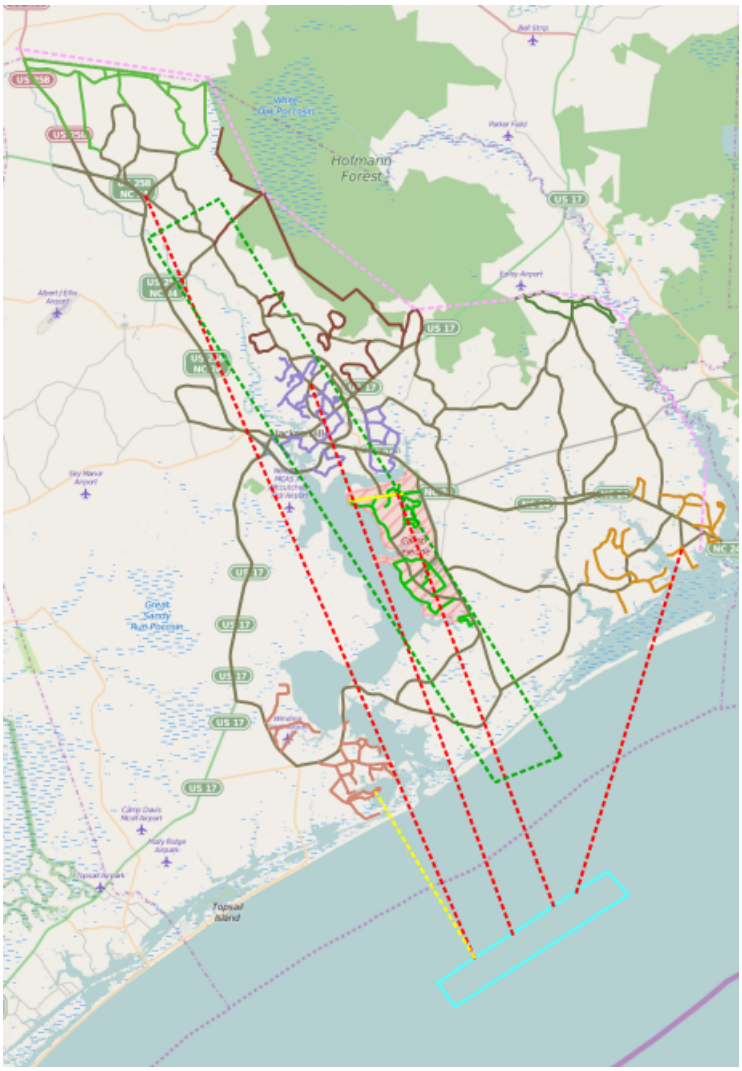


Figure 3.5. The Bold Alligator mobility scenario base implementation map. Source: [3].

While Bold Alligator's mobility concept is similar to the Omaha scenario, it implements more node types with greater mobility variation. In addition to different mobility, Bold Alligator also implements radios with longer transmission ranges. During testing, Bold Alligator, like Helsinki and Omaha, makes use of ten variations of node mobility implemented on the same map in the form of ten mobility files. Table 3.3 is taken from [4] and describes the radio parameters used in the Bold Alligator scenario.

Table 3.3. Bold Alligator scenario parameters. Adapted from [4].

Parameter	**Values**
Simulation Duration	24 hrs
Simulator Seed	717, 718
Warmup Time	10800 s
Timestamp Resolution	0.1 s
Beacon Interval	5 s
Number of Marines	70
Number of Humvees	20
Number of Drones	2
Number of Helicopters	8
Number of LCACs	2
Number of Ships	3
Base Radio Bandwidth	12, 24, 36, 54 Mbps
Humvee Radio Bandwidth	54 Mbps
Drone Radio Bandwidth	6 Mbps
Ship Radio Bandwidth	54 Mbps
Base Radio Transmit Range	100 m
Humvee Radio Transmit Range	3000 m
Drone Radio Transmit Range	3000 m
Ship Radio Transmit Range	10000 m
Marine Node Buffer Size (MB)	5
Drone Node Buffer Size (MB)	5
Humvee Buffer Size (MB)	50
LCAC Buffer Size (MB)	50
Helo Buffer Size (MB)	50
Ship Buffer Size (MB)	500
Marine Message Size	250 - 500 KB
Humvee Message Size	0.5 - 1.0 MB
Ship Message Size	0.5 - 1.0 MB
Marine Message Rate	1 / 5 - 10 s
Humvee Message Rate	1 / 10 - 20 s
Ship Message Rate	1 / 25 - 35 s
Message TTL	5 hrs
Protocols	Epidemic, Vector, Centroid, GAPR, GAPR2, GAPR2a, QGeo

3.3.4 Swarm

The Swarm scenario presents a unique testbed for mobile nodes because it uses real-world recorded data, without significant modification or generalization. The initial implementation was built by Pospischil using GPS logs from recorded flight data in 2015 at NPS when the Advanced Robotics System Engineering Laboratory (ARSENL) group flew multiple flights of drones in swarms of various size [5]. The swarms were composed of aerial nodes and a ground station that acted as an arbiter for the swarms. Each flight test simulated an adversarial pair of sub-swarms of equal sizes. While this scenario illustrates the potential military use of the swarms, the reality is that the communications infrastructure did not differentiate between the opposing swarms. The network for a scenario of two six-drone swarms was a 13-node network composed of 12 aerial nodes and one ground station. Because the drones are aerial nodes by nature, they do not rely on map features, and all nodes have the same flight speed. However, the initial ns3 implementation translated the real-world GPS logs into ns2 format using the `ConstantPositionMobilityModel` so that a node was assigned a position at some *time*. Then at the next recorded GPS log for that node, an ns2 statement was implemented to automatically move the node to its next recorded position, with no continuous motion correlating the two positions.

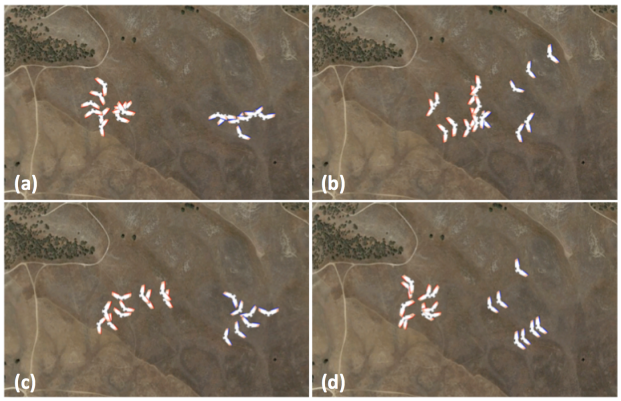


Figure 3.6. The Swarm mobility scenario base implementation graphic. Source: [5].

ns3 contains many built-in mobility models; however, none of them are directly map based. All of the built-in models rely on some type of random movement, constant position (discrete movement), or constant velocity vector (linear movement). In order to implement a real-world mobility model, an external format must be used. ns3 supports the ns2 mobility

model. An ns2 mobility file contains single-line commands pertaining to specific nodes, as described in the ns3 documentation [32]. The ns2 mobility format allows three canonical statements, from which the ns3 class `Ns2MobilityHelper` translates simulator mobility events. The ns3 model library lists the following command types:

- $node set X_ x1 : Sets the local [X_, Y_ or Z_] position of *node* to [x1, y1 or z1].
- $ns at $time $node setdest x2 y2 speed: Commands *node* to move at *time* toward the location described by [x2, y2] with velocity *speed*.
- $ns at $time $node set X_ x1: Commands *node* to immediately change its current [X_, Y_ or Z_] position at *time* to [x1, y1 or z1].

This study introduces a new `Ns3MobilityHelper` format that modifies the built-in ns2 mobility format used by the previously mentioned scenarios. ns3's built-in `Ns2MobilityHelper` can assign static x, y and z positions, but can only use the "set-dest" command format to assign an x and y component for the destination, as described previously. The new `Ns3MobilityHelper` implements a z-component for "setdest" commands that schedule a node beginning continuous motion toward the assigned destination. With the z-component enabled in the `Ns3MobilityHelper`, the GPS logs were converted to the new ns3 format and form the basis for mobility for the Swarm scenario. Following Pospischil's implementation, swarms are implemented in 6v6 (12-node), 10v10 (20-node), 15v15 (30-node) and 25v25 (50-node) groups. Subsection 3.3.5 provides more detailed information and analysis of the impact of the `Ns3MobilityHelper` implementation on swarm mobility behavior.

The initial Swarm implementation used four applications based on the ns3 built-in `OnOffHelper`, which implements an on/off timer for application message generation. This application model worked for MANET testing, as recorded in [5]; however, for DTN protocols, the `DTNHelper` class is necessary for application compatibility. Another contribution of this study is the implementation of the `DTNHelper` at the application level which allows testing of DTN protocols in the Swarm scenario. Testing of the Swarm scenario is conducted with four base mobility files corresponding to the four swarm sizes previously listed. Table 3.4 describes the radio parameters used in the Swarm scenario.

Table 3.4. Swarm scenario parameters

Parameter	**Values**
Simulation Duration	25 min
Simulator Seed	1
Warmup Time	0 s
Timestamp Resolution	0.001 s
Beacon Interval	5 s
Number of Drones	12 - 50
Radio Transmit Gain (Relative)	0, 3, 6, 9, 12, 20
Radio Transmit Range	550 m
Node Buffer Size (MB)	5MB
Message Rate	1 / 1 - 2 s
Message Size	20b, 56b
Message TTL	N/A
Protocols	Epidemic, Vector, Centroid, GAPR, GAPR2, GAPR2a, QGeo

3.3.5 Swarm Case Study and Analysis

In order to validate the updated swarm mobility model, a test was performed that compared recorded mobility logs as well as actual network performance for each of the implemented swarm sizes. This section discusses the specific method of translation for the mobility baseline datasets, parameters of the validation experiment, and describe the results of the experiment.

Dataset Translation Methodology

This study modified the raw datasets used in Pospischil's original implementation [5] by using a Python script to take each line of data and convert it to the ns3 format described in Section 3.3.4. Since each raw data file had thousands of lines of recorded location data, it was necessary to automate this process rather than doing it by hand. The first iterations of this automation revealed that there were multiple lines in each file with inconsistent GPS data. Further explanation is revealed in Section 3.3.5, but as an example illustrating the inconsistent data, the following list reveals a GPS error. Each line has the following format: $time, swarm\_number, node\_number, x\_location, y\_location, z\_location$.

- 1488250647.969,2,20,12043819.229455,4186523.341756,0
- 1488250648.069,2,20,47.58144,18.675451,275.38

The two lines shown above were taken from the raw 10v10 swarm data file. These lines state that node 20 moved from $(x, y, z) = (12043819.229455, 4186523.341756, 0)$ to $(x, y, z) = (47.58144, 18.675451, 275.38)$ in 0.9 seconds exactly. This means that node 20 moved 12750657.11664923 meters in 0.9 seconds, which is equivalent to a speed of 14167396.796276921 m/s. This is 4% of the speed of light, which is approximately 30,000x the speed of the earth's rotation at the equator. This is clearly an error in the recorded data. What likely happened when the drone was recording GPS logs is that the GPS receiver was recycled or calibrated between the two recorded times.

With the above explanation of the dataset errors in mind, the automated script used to build the ns3-formatted datasets was written to skip lines from the raw files that revealed this GPS error. Therefore, there are slightly less lines in the ns3-formatted mobility data files than there are in the raw files. Additionally, times from the raw format were offset by the first time, in order to make the starting time of the ns3-formatted files start at time 0. The raw files were still tested without modification against the translated ns3-formatted files.

In order to capture the effect of the employed base mobility data files, ns3's `MakeBoundCallback` function was used to track node course change movements. The `MakeBoundCallback` function is effectively a listener that operates in ns3's simulation background and can be configured to listen to various parameters, then making a callback to a function written inside the running simulation file. For the purposes of swarm mobility tracking, the callback function employed in the swarm simulation records useful information for each course change detected. To explain further, every time a node changes location, the callback function records the time, nodeID, location and velocity of that node.

Mobility Study Test Parameters

The mobility test was performed for the 12, 20, 30 and 50 node swarms using the Destination-Sequenced Distance Vector (DSDV) routing protocol. DSDV was chosen as the routing protocol because it was previously implemented in [5], so network performance comparison has an independently recorded baseline, and simulations running MANET protocols tend to run faster than those using DTN protocols. In addition to the routing protocol, the

applications implemented for this test were the heartbeat and pose applications (no broadcast applications), radio gain was set to 6 and 12 for diversity, and mobility files used were the original raw (cleaned) datasets and translated ns3-formatted datasets for comparison. All other experimentation parameters such as simulation duration, warmup time, beacon interval, etc. were the same as the parameters described for the swarm scenario testing at the end of Section 3.3.4.

Mobility Test Analysis and Findings

For analysis of the effects of the employed mobility models, a Python script was written to parse the output file from the callback function described in Section 3.3.5. The callback function for each simulation run outputs a movement log file corresponding to the number of nodes and base mobility data file used. The Python analysis script written for the mobility analysis compared the two movement log files with the same number of nodes, but different base mobility data file. The output from the Python analysis script is illustrated in Table 3.5.

Table 3.5. Recorded mobility comparison

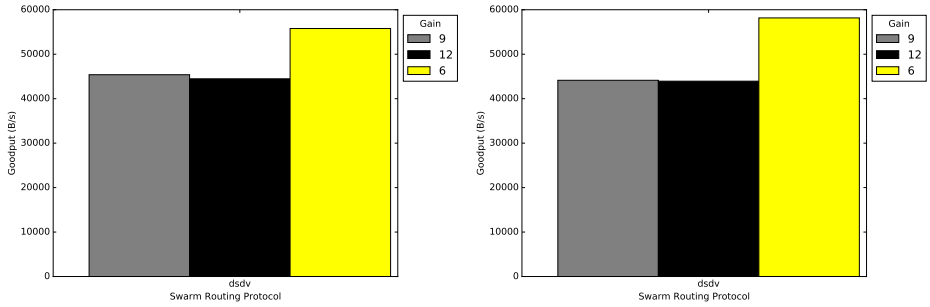
Data Recorded	**6v6**	**10v10**	**15v15**	**25v25**
Total recorded location times	83362	243998	341768	289365
Matched location times	42965	97592	136338	117460
Percent matched (%)	54%	40%	40%	41%
Raw average difference in recorded location (m)	3.447	2682690.927	10.028	534915.345
Number of GPS errors (difference > 100m)	0	20546	0	5308
Average difference in location (GPS errors removed) (m)	3.447	9.629	10.028	20.213
Max difference in location (m)	13.849	12750682.480	35.628	12750682.520
Min difference in location (m)	0.0	0.0	0.0	0.0
Number of exact location matches	15037	786	12522	2041

Table 3.5 shows the recorded data from the mobility output log comparison. Difference in location between recorded node locations when using the old mobility base data file versus the ns3 mobility format data file was only calculated for records with matching times for the same node. As Table 3.5 illustrates, roughly 40% of recorded locations actually had matching times for the same node. This is significant because it is impossible to compare the location of a node unless the location was recorded by the course change callback function at the exact same time. The percent matched field in Table 3.5 gives the baseline of the relative amount of data compared in this experiment, compared to the total available data.

The raw average difference in recorded location was calculated with a cartesian distance calculation. The average location difference for the 20 and 50 node swarms clearly had a large divergence due to locations recorded from GPS errors. In testing, the number of GPS errors was the same when the threshold was set to 100m as it was when the threshold was set to 100,000m. For this reason, it is clear that the recorded locations came from the GPS error described in Section 3.3.5. Maximum and minimum recorded location difference was included to illustrate the range of location differences. The recorded minimum location difference of 0.0 for all scenarios illustrates that there were always locations recorded at the same time for the same node in every scenario that were the exact same location. The last row in Table 3.5 shows the number of times that recorded locations were exactly the same in each scenario.

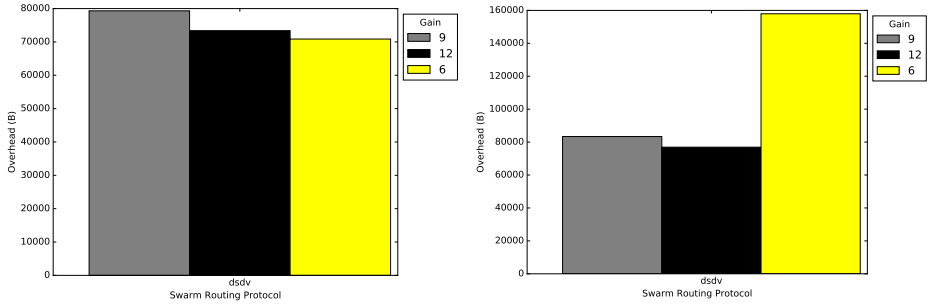
Another useful measure of the effect of the modified mobility base files used for the swarm scenario is the change in networking performance while using the modified base files. Since DSDV was used in testing the mobility implementations, the network trace files output by each scenario are not reflective of any DTN performance; however, as mentioned in Section 3.3.5 the original mobility model implemented in [5] tested MANET protocols, so testing the new mobility model with DSDV is useful for comparing against Pospischil's baseline output. Metrics collected for this testing are: MDR, goodput, overhead and latency. A general description of each of these metrics can be found in Section 3.4.

As a brief overview of the results, Figures 3.7 and 3.8 show results for the 12-node scenario using DSDV with radio gain set to 6 and 12. The similarity of the results, and in particular the similarity in the trends across almost all of the results are a good indicator that the performance of the routing protocol is not greatly effected by the change in mobility model implementation. The collection of output charts and plots describing the networking performance change incurred when using the modified mobility base files can be found in the swarm mobility appendix. Generally, each of the swarm scenarios shows consistent network characteristic outputs, and the data represented in the graphs show consistent trends.



Constant Position Mobility Model ns3 Mobility Model

Figure 3.7. 6v6 goodput



Constant Position Mobility Model ns3 Mobility Model

Figure 3.8. 6v6 overhead

With the modified mobility model showing consistent trends across all networking results and revealing some significant differences across recorded location metrics, it is useful to note that the principle employed in the modified mobility model is the interpolation of location commands. In particular, since the `Ns3MobilityHelper` gives nodes movement commands, ns3 simulations using this mobility model interpolate node locations, rather than assigning constant positions and discrete location changes. This means that the nodes using this mobility model, in particular for the swarm scenario tested, nodes exhibit more realistic movement behavior.

3.4 Metrics

This section discusses the metrics used for analysis across all testing. MDR, overhead, goodput, latency and hop count were used, and each simulation was run multiple times in

order to acquire statistically significant data. Since ns3 can output network trace data in a format that is effectively comparable to a tcpdump, and rather than reading through the tens of thousands of lines of output for every single scenario, these metrics were analyzed through the employment of a parsing script written in the Python language. All analysis statistics and graphs were generated by this script, using multiple Python libraries to include numpy for statistical analysis and matplotlib for plot generation.

While power consumption was not analyzed, it is desirable for DTN protocols to have low power consumption because mobile nodes usually need to manage power consumption efficiently across the various levels of node operation. It is noteworthy that routing protocols that transmit messages across more hops inherently use more power across the network. From a per-node basis this is also relevant because more hops-per-message implies higher total number of nodes transmitting that message than would otherwise be transmitted; therefore, each node inherently uses more power throughout the given scenario. It can be concluded then that protocols with a higher overhead are generally less power-efficient.

3.4.1 MDR

MDR is a calculation of the delivery success rate of a given protocol. It is a useful metric because it describes how well a protocol performed in terms of getting messages to their destination, regardless of e2e connectivity. MDR is calculated by Equation 3.1 on a network-wide basis. It can be calculated on a per-node basis, but as it relates to this study, it makes more sense to view MDR as a network-wide metric. Since MDR is a ratio, there is no unit of measurement for the calculation.

$$\text{MDR} = \frac{\text{\# Messages delivered}}{\text{\# Messages transmitted}} \tag{3.1}$$

3.4.2 Message Overhead

Overhead is a general term that can describe many levels of extraneous data. It is often used to describe data or time required for connectivity, data delivered that is not relevant to the application being measured (control messages), or data that is replicated. For the purpose of this study, Message Overhead is measured as replicated data messages. Only messages

sent with a payload, or application-level data, is considered useful traffic; control messages are not considered. Message Overhead is measured by Equation 3.2.

$$\text{Message Overhead} = \frac{\text{Total packets transmitted} - \text{Total packets delivered}}{\text{Total packets delivered}} \tag{3.2}$$

3.4.3 Goodput

Goodput, like the more commonly used throughput, is a measure of total useful data delivered during protocol operation. The difference between throughput and goodput is that throughput looks at total data received while goodput removes duplicate messages and retransmissions from its calculation. Goodput is measured on a per-node basis in Bytes-per-second (B/s). For this study, Goodput is measured as an average of all nodes' goodput and is described by Equation 3.3.

$$\text{Average Goodput} = \frac{\text{Average unique bytes received per node}}{\text{Average delivery time per message}} \tag{3.3}$$

3.4.4 Latency

Latency is a measure of delay for a given message delivery. It is a useful metric in studying DTN protocols because it gives helpful insight to the protocol's efficiency in delivering application-level messages. Many applications require messages to be delivered quickly, so protocols that induce high latency are not helpful for those types of applications. Latency is typically calculated on a per-message basis, but for this study, is measured as an average across all messages. Latency is calculated as:

$$\text{Time of message delivery} - \text{Time of message transmission} \tag{3.4}$$

3.4.5 Hop Count

Hop Count is the record of how many hops a single message took from transmission to arrival at its assigned destination. It is a useful metric in understanding the effectiveness of node-

to-node routing decisions, especially when viewed alongside latency and/or throughput. For the purposes of this study, hop count is taken as an average measurement across all nodes in the network. The average hop count is a floating point value, and the units are hops per message.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Results and Analysis

This Chapter discusses the results of testing data, and in-depth analysis of the gathered data in a summarized format. For the purpose of compressing the large amount of raw data analyzed, charts and tables are used to illustrate the various routing protocols' performance for each evaluated metric. Each section contains a table that illustrates the associated values with the various metrics for each protocol tested, as well as charts and/or plots that describe the performance of each protocol for a specific metric in greater detail. A brief comparison of this study's analyzed data with Mauldin's findings [4] for similar experimental parameters, as illustrated in Section 3.3.1, reveals that the protocols' relative performance is generally matched for the Helsinki scenario. In other words, after an analysis of the obtained data for the Helsinki tests, it is reasonable to have confidence in the data because it matches the trends of data found in previous studies with similar experimental parameters. This proves the fidelity of the findings of this study, and is illustrated throughout the Helsinki analysis. Because of the consistency, the Omaha and Bold Alligator performances are not directly compared, but left for the reader to make the direct comparison of protocol performance between the studies.

4.1 Helsinki Results

Table 4.1 represents the summarized data for the Helsinki scenario. Each column represents the data for a specific protocol. The average unique messages transmitted and delivered describe the average unique message ID numbers that were recorded across each simulation run for all transmissions and deliveries. The average unique messages transmitted for each protocol are very similar but not the same. The reason these values are not the same, despite running on the same application framework in the simulation scenario is that each protocol has different forwarding schemes, and when new messages are generated, they are put in a node's buffer for later transmission. Therefore, the messages generated will be the same for each protocol, and is dependent on the simulation, but the unique messages transmitted illustrates the protocol's behavior of determining which messages are released from its queue for transmission, at each node interaction. The MDR row represents the

average of the calculated MDR value for each simulation run, not the result of dividing the unique received messages by the unique transmitted messages. Further discussion of the MDR is provided in Subsection 4.1.1. The average total transmissions and receptions illustrates each protocol's message replication traffic. The average data transmitted and received rows correspond to the calculation of each unique message multiplied by its size in bytes, summed across all transmissions and destination receptions, respectively, as recorded throughout each simulation run. The average goodput is the average of all recorded goodput values. Similar to the MDR values, the average goodput values are an average of averages, so they will not directly correspond to a calculation done on the average data received, divided by the total simulation time for the Helsinki scenario. Subsection 4.1.3 elaborates this discussion. Finally, the average latency mean, median, and max, as well as the average hop count mean, median, and max were calculated for each simulation run, and Table 4.1 describes the averages of those values in the correspondingly named rows. The hop count values for Epidemic, Vector, and Centroid are not accurate, but approximate the hop count values based on adjusted averages. Elaboration on the errors with the hop counts for those protocols is provided in Subsection 4.1.5.

Table 4.1. Helsinki results

Parameter	Epidemic	Vector	Centroid	GAPR	GAPR2	GAPR2a	QGeo
Average Unique Messages Transmitted (K)	1.39	1.29	1.02	1.3	1.24	1.21	1.14
Average Unique Messages Delivered (K)	0.49	0.91	0.8	1.03	0.91	1.03	0.86
Average MDR	0.35	0.71	0.78	0.79	0.73	0.85	0.76
Average Total Transmissions (K)	33,846.37	5,338.44	10,690.24	50,414.41	4,700.21	11,939.86	4,065.13
Average Total Receptions (K)	31,516.52	4,963.97	10,171.71	47,582.48	4,359.54	11,384.12	4,017.05
Average Data Transmitted (GB)	69.32	10.93	21.89	103.25	9.63	24.45	8.33
Average Data Received (GB)	64.55	10.17	20.83	97.45	8.93	23.31	8.23
Average Overhead (GB)	69.31	10.93	21.89	103.25	9.62	24.45	8.32
Average Goodput (B/s)	23.74	44.38	38.69	50.07	44.31	50.05	41.92
Average Latency Mean (s)	1,064.07	4,658.9	3,660.86	4,315.26	4,706.72	3,783.89	2,569.49
Average Latency Median (s)	929.88	3,869.43	3,160.76	3,367.91	3,850.07	2,908.26	574.2
Average Latency Max (s)	5,457.49	17,147.81	15,725.02	17,386.43	16,903.22	17,135.24	17,090.99
Average Hop Count Mean	5.0	4.0	4.7	1.05	1.0	1.0	1.0
Average Hop Count Median	(4.65)*	(3.0)*	(3.55)*	2.0	1.05	1.25	1.0
Average Max Hop Count	(1.0)*	(1.0)*	(1.0)*	6.25	4.25	4.65	4.7

4.1.1 MDR

Figure 4.1 represents the calculated MDR averages for the Helsinki scenario. The average MDR is the average of all of the MDR values calculated across each simulation run. For that reason, with regard to Table 4.1, the value resulting from the division of the average unique messages delivered by average unique messages transmitted is not exactly the same because the MDR values represented are an average of previously calculated averages. For

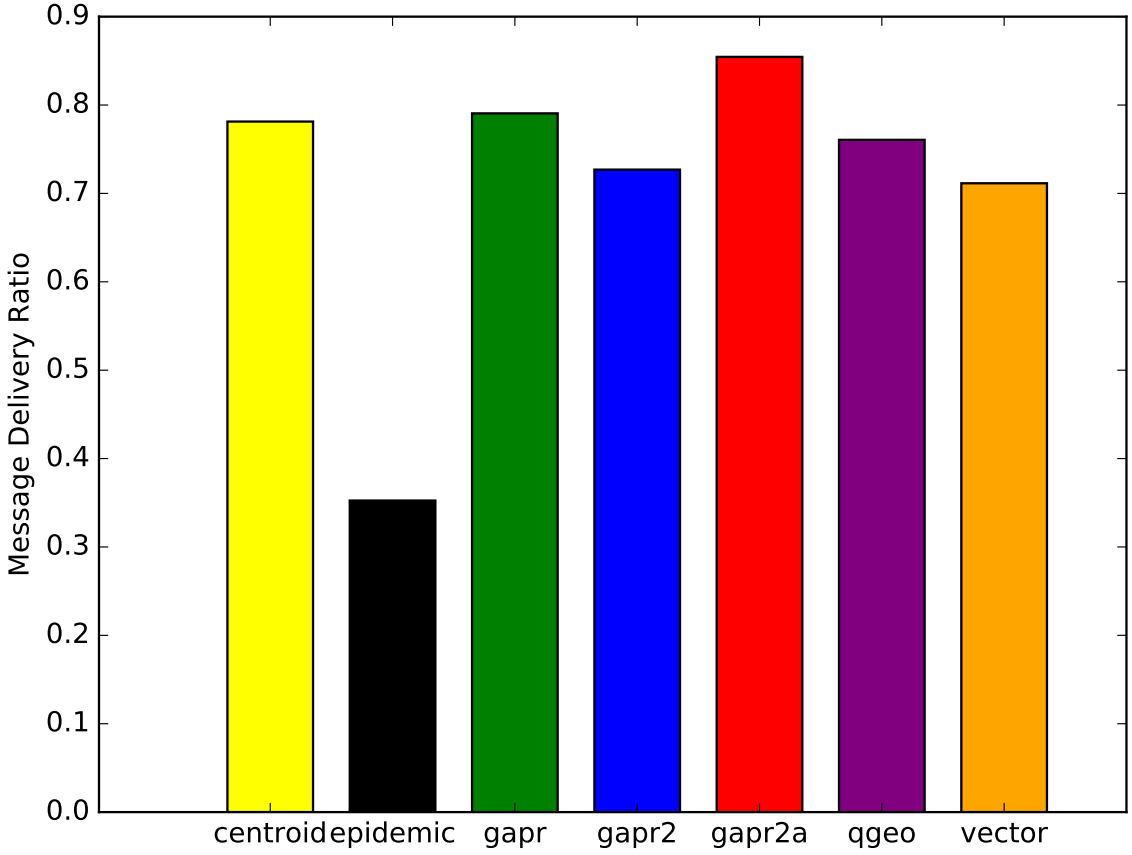


Figure 4.1. Helsinki MDR

the Helsinki scenario, the represented MDRs are very close to the value of average delivered messages divided by average transmitted messages.

One peculiarity about the Helsinki results is that Epidemic shows such a low MDR. This result is consistent with Mauldin's findings [4], so it is not unexpected, but since Epidemic is a flooding-based protocol, its main goal is to get all of the messages to their destinations, regardless of the cost. It follows that Epidemic should have a high MDR. Nonetheless, the other protocols provide a higher MDR, which is likely because of their queue control schemes. Since Epidemic floods all messages and has no forwarding limitation mechanism, while all of the other protocols do have methods for controlling message forwarding, Epidemic generally delivers fewer unique messages, while transmitting as many, or more than the other protocols. This is consistent across the majority of the scenarios. In addition to Epidemic's performance, GAPR2a has the highest MDR, while QGeo's performance is

generally comparable to the other protocols.

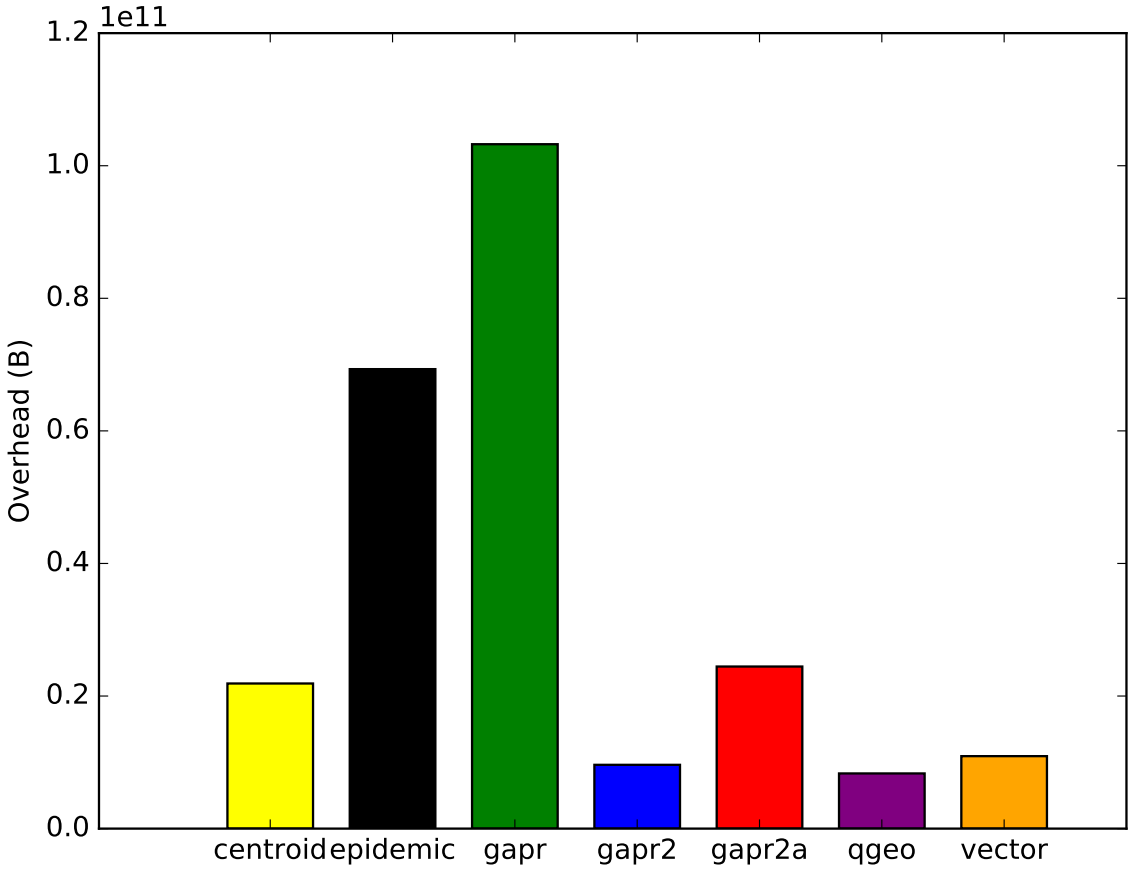


Figure 4.2. Helsinki overhead

4.1.2 Overhead

Figure 4.2 depicts the Helsinki overhead findings. In terms of message replication overhead, QGeo offered a relatively low value. This is a significant finding for QGeo, as it does not implement the message forwarding limitations that Vector, Centroid, or the corresponding GAPR protocols implement. Instead, QGeo controls which messages are transmitted by viewing each message's destination, and the corresponding Q-values. The other protocols generally use the queue order to determine which messages are transmitted, instead of a per-message determination. Notably, comparison of the other protocols with Mauldin's findings [4] shows consistent relative results, with GAPR and Epidemic having the highest overhead, while GAPR2 and Vector were the lowest. It is notable that in terms of

message replication overhead, there is somewhat of a pairing of like protocols, excluding QGeo. Epidemic and GAPR have the highest overhead, and they do not implement a message forwarding limitation for their queue, during a given interaction. Vector and GAPR2 have similar overhead values, and GAPR2 implements Vector's forwarding limitation mechanism. Similarly, Centroid and GAPR2a have similarly paired overhead values, and GAPR2a implements Centroid's message forwarding limitation calculation. An important finding from this comparison is that when comparing message forwarding limitation schemes, Vector's calculation limits message replication overhead better than Centroid's, or having none. Additionally, QGeo's per-message limitation performs similarly to Vector's calculation.

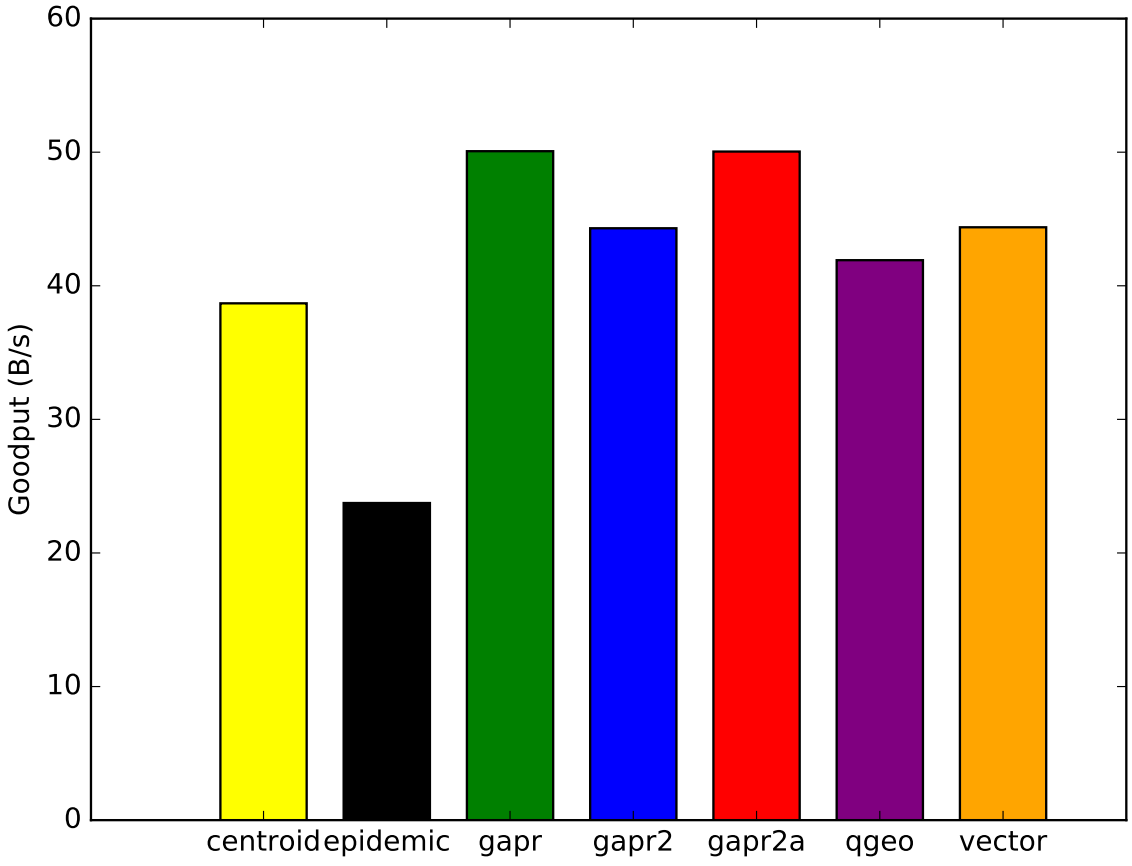


Figure 4.3. Helsinki goodput

4.1.3 Goodput

Figure 4.3 describes the Helsinki goodput results. The values represented in the figure correspond to the average goodput values depicted in Table 4.1. However, these values do not correspond to the value determined by calculating the average data received divided by the total application running time in the simulation. The Helsinki scenario applications ran for 42200 seconds. Dividing average data received by 42200 seconds does not give the same average goodput value represented in Figure 4.3 because, as with MDR, the average goodput is an average of averages. An additional factor contributing to the difference in values is that goodput measured as an average can diverge greatly from goodput values recorded for a certain interval within a simulation. Just like a protocol's average throughput does not represent the constant rate of transmission of the protocol, a protocol's average goodput does not represent a constant rate of data delivery. Instead, average goodput represents the overall average of data delivered to the required destination on a per-message basis throughout the scenario, divided by the amount of time each unique message was in flight. Therefore, average goodput is not depicting a constant rate of delivery, but the average of each unique message's size divided by how long it took to get to its destination, averaged for all unique message deliveries in the simulation. If the average goodput is multiplied by the entire simulation time, the total data delivered is orders of magnitude higher than the result, because it is measured for unique messages, and only for the time they are in flight. This is consistent with the data in Table 4.1.

Since goodput is calculated based on unique messages delivered, it follows that the goodput results should have similar trends to the MDR results. A comparison of the two plots shows that they do, in fact, have many similarities. Namely, Epidemic performs the worst in both, while the GAPR protocols generally perform the best, with QGeo slightly worse. The goodput performance can be attributed to the same buffer control and message forwarding mechanisms employed by each protocol, as the buffer control is determined based on total byte size, rather than number of messages alone. Goodput was not used in Mauldin's or Killeen's works, which are the most comparable to this study's work, so no useful direct comparison is available for the goodput charts; however, the similarity in the MDR results validates the data.

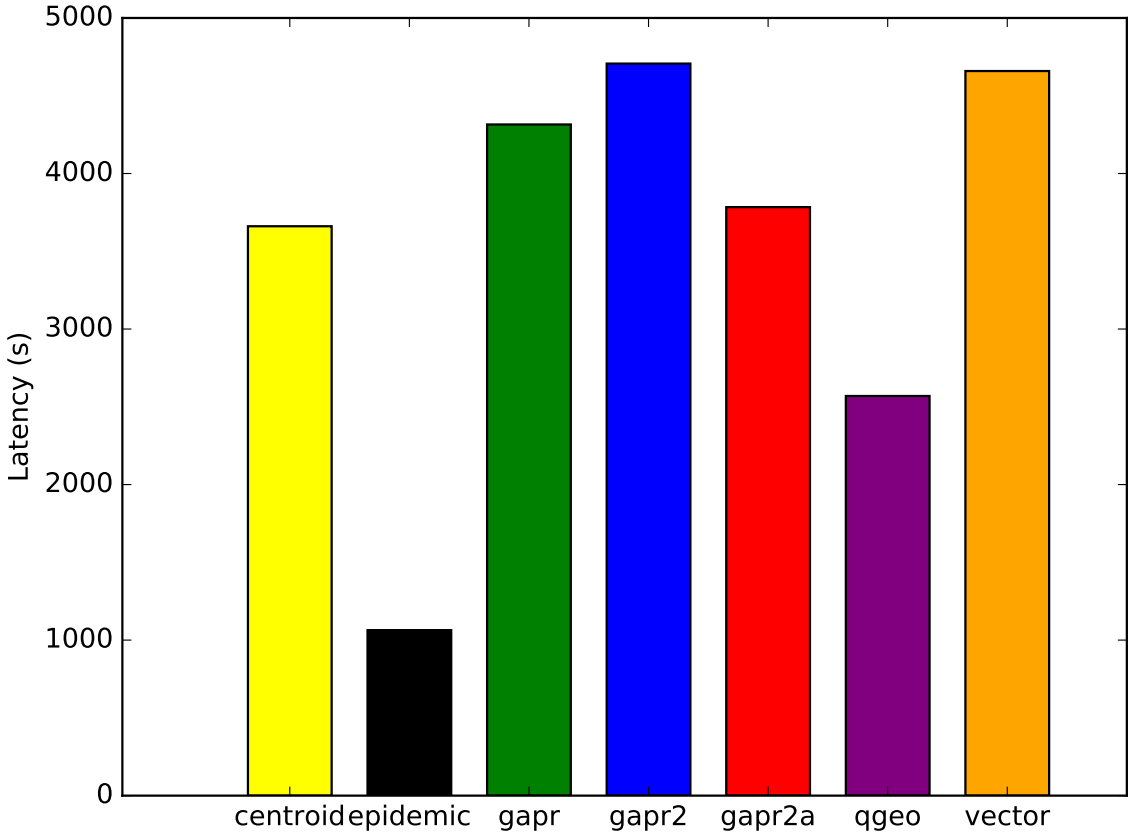


Figure 4.4. Helsinki latency

4.1.4 Latency

Figure 4.4 represents the Helsinki latency results. As expected, Epidemic performs the best of the protocols, likely due to its flooding mechanism. As with Overhead, there is a distinct correlation between Vector and GAPR2's performance, and between Centroid and GAPR2a's performance, due to their message forwarding limitation mechanisms. These mechanisms ensure that nodes' buffers are managed through limiting message transmissions. However, unlike with Overhead, GAPR and Epidemic show a very different performance. Though GAPR does not implement a message forwarding limitation, it forwards messages based on interaction probability, as with all of the GAPR protocols. This means that it forwards messages based on prioritized order after determining the probability of delivery for each message. These results show that the forwarding prioritization mechanism slows down the delivery of messages significantly, as compared to a pure flooding mechanism, as in Epidemic. Alternatively, QGeo mitigates some of this delay by learning which messages

are likely to be delivered based on Q-values, and only forwarding those messages at a given interaction. These results show that the learning mechanism aids to deliver messages faster than a queue-based limitation. Nonetheless, Epidemic's flooding mechanism outperforms all of the other protocols. As in Mauldin's findings [4], Vector and GAPR2 have the highest latency, while Epidemic has the lowest.

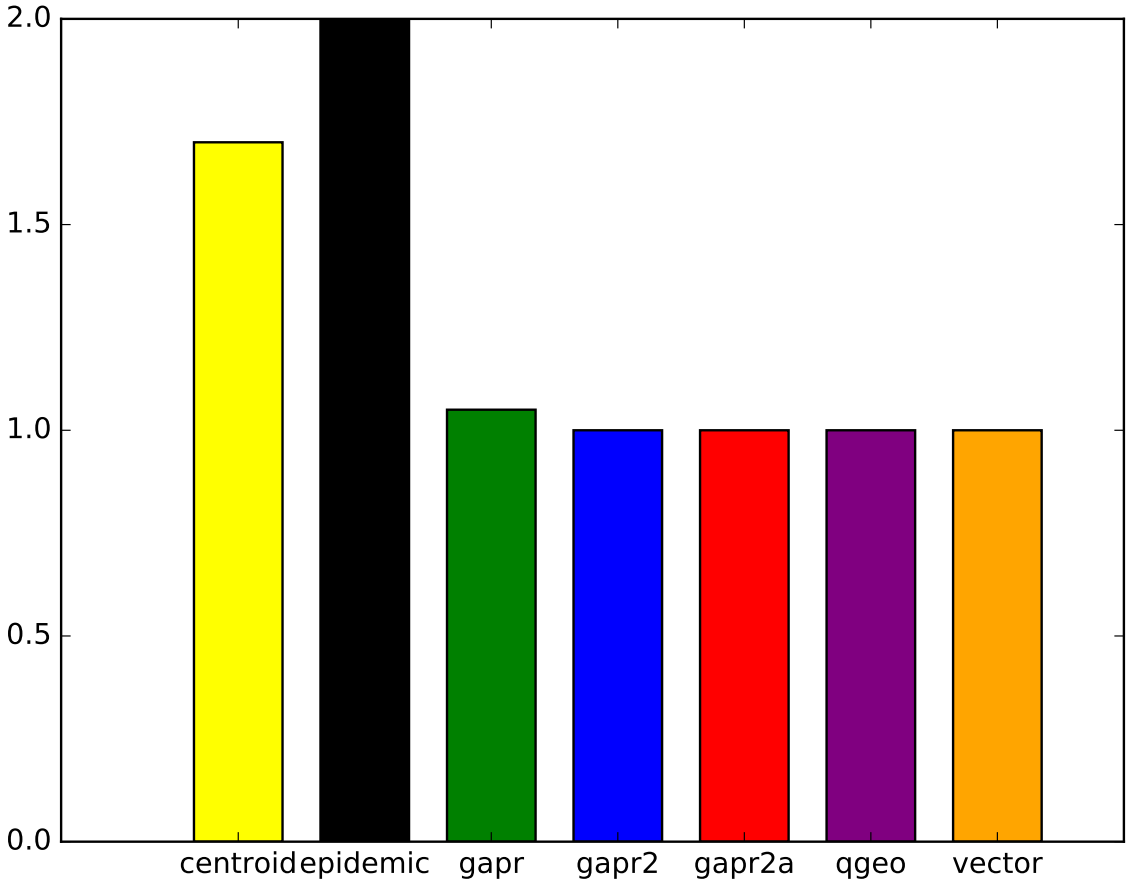


Figure 4.5. Helsinki hop count

4.1.5 Hop Count

Figure 4.5 represents the Helsinki hop count measurement. The GAPR, GAPR2, GAPR2a, and QGeo hop count values were accurately measured; however, the values for Epidemic, Vector, and Centroid were not correctly recorded. The ns3 implementations of Epidemic, Vector, and Centroid use the max allowable hop count value (hop count limit) as the hop count value to record in the IP-layer trace callback, which are the trace files that this study

used to measure the recorded metrics. Mauldin's study used a different trace callback at the application layer [4], that subtracts the max hop count from the current header's hop count value for each message, to report the hop count accurately in the application layer trace files. In order to approximate the corresponding hop count values for Epidemic, Vector, and Centroid for this study, the calculation was performed after the trace files were parsed, so the maximum hop count value was subtracted from the average hop count values recorded, rather than from each line of the trace file as done in the application layer trace file. Therefore, while the Epidemic, Vector, and Centroid average hop count values are not entirely accurate, they do approximate the correct values. The mean, median, and max hop count values recorded in Table 4.1 for Epidemic, Vector, and Centroid are calculated based on the approximate averages, rather than on a per-message basis, unlike for QGeo and the GAPR protocols, comparison across the protocols should be informed by those approximations. The reader is referred to Mauldin's study [4] for a more accurate hop count measurement for Epidemic, Vector, and Centroid.

Keeping in mind the issue with the calculated hop count values for Epidemic, Vector, and Centroid, and after comparing their values to Mauldin's study [4], it is clear that QGeo significantly reduces the number of hops that messages require, on average, to reach their destination. This likely indicates, along with Goodput and MDR measurements, that QGeo holds messages longer than other protocols, only forwarding them at the most opportune times.

4.2 Omaha Results

The Omaha results are mostly consistent with the trends shown in the Helsinki results. The Omaha scenario employs a lower radio bandwidth and lower total number of nodes than the Helsinki scenario, which has a significant impact on the various protocols' performance for each recorded metric. Table 4.2 shows the averaged aggregate results for the Omaha scenario. Like the Helsinki table, Table 4.2 is separated into columns to represent each protocol's results. The MDR average of averages representation issue is the same for the Omaha scenario as for the Helsinki scenario, as described in Section 4.1. The average unique messages transmitted and received represent unique message IDs recorded, while the average total messages transmitted and received correspond to the message replication per protocol in the Omaha scenario. Average goodput describes the average of measured

goodput across each simulation run for the Omaha scenario, while average data transmitted and received is the average of the number of messages transmitted and received, multiplied for each message by the byte-size of each message. As in the Helsinki scenario, latency mean, median, and max is measured as the average end-to-end delay in delivering each message. Finally, mean, median, and max hop count was measured the same way as in the Helsinki scenario, and has the same protocol implementation issues for hop count in Epidemic, Vector, and Centroid as in Helsinki.

Table 4.2. Omaha results

Parameter	Epidemic	Vector	Centroid	GAPR	GAPR2	GAPR2a	QGeo
Average Unique Messages Transmitted (K)	1.28	1.19	0.87	0.96	0.94	0.83	0.48
Average Unique Messages Delivered (K)	0.25	0.27	0.16	0.29	0.28	0.28	0.21
Average MDR	0.19	0.23	0.19	0.3	0.3	0.34	0.43
Average Total Transmissions (K)	21,869.39	20,478.68	9,698.67	57,061.3	33,081.48	32,685.46	15,826.71
Average Total Receptions (K)	21,003.41	19,992.32	9,532.19	54,797.54	32,382.63	32,153.89	14,654.81
Average Data Transmitted (GB)	44.79	41.94	19.86	116.86	67.75	66.94	32.41
Average Data Received (GB)	43.01	40.94	19.52	112.23	66.32	65.85	30.01
Average Overhead (GB)	44.79	41.94	19.86	116.86	67.75	66.94	32.41
Average Goodput (B/s)	11.95	13.34	8.0	14.11	13.46	13.68	10.08
Average Latency Mean (s)	1,588.37	4,116.47	2,534.47	4,615.68	4,757.48	3,423.57	2,045.67
Average Latency Median (s)	1,091.58	2,264.08	718.6	2,756.29	3,041.12	1,519.92	322.22
Average Latency Max (s)	10,265.98	17,337.15	16,135.45	17,646.31	17,415.41	17,194.83	15,946.44
Average Hop Count Mean	4.3	9.25	8.75	1.2	1.0	1.0	1.0
Average Hop Count Median	(3.55)*	(2.77)*	(2.48)*	1.75	1.1	1.45	1.05
Average Max Hop Count	(1.0)*	(1.0)*	(1.0)*	6.6	5.35	5.25	3.55

4.2.1 MDR

Figure 4.6 displays the Omaha MDR output. Notably, the actual MDR values are much lower in the Omaha scenario than the Helsinki scenario. Every protocol has a lower MDR in Omaha, and on average, the protocols' (with the exception of Epidemic) MDR in the Omaha scenario is about half that of the Helsinki scenario, for each protocol. This speaks to the reduced bandwidth. In the Omaha scenario, nodes interact less often and have a lower available bandwidth, which means that when they do interact, they can forward fewer messages. Unlike in the Helsinki scenario, the protocols show a slight grouping in the Omaha scenario. QGeo and the GAPR protocols all have relatively improved performance over Epidemic, Vector, and Centroid, whereas in the Helsinki scenario, they were all relatively grouped together, except for Epidemic. This speaks to the buffer management techniques of the GAPR protocols and QGeo. The reduced opportunity to forward messages in the Omaha scenario implies that in order to successfully propagate messages to their destinations, nodes must opportunistically forward messages. QGeo and the GAPR protocols have buffer mechanisms for choosing which messages (Q-values and buffer reorganization

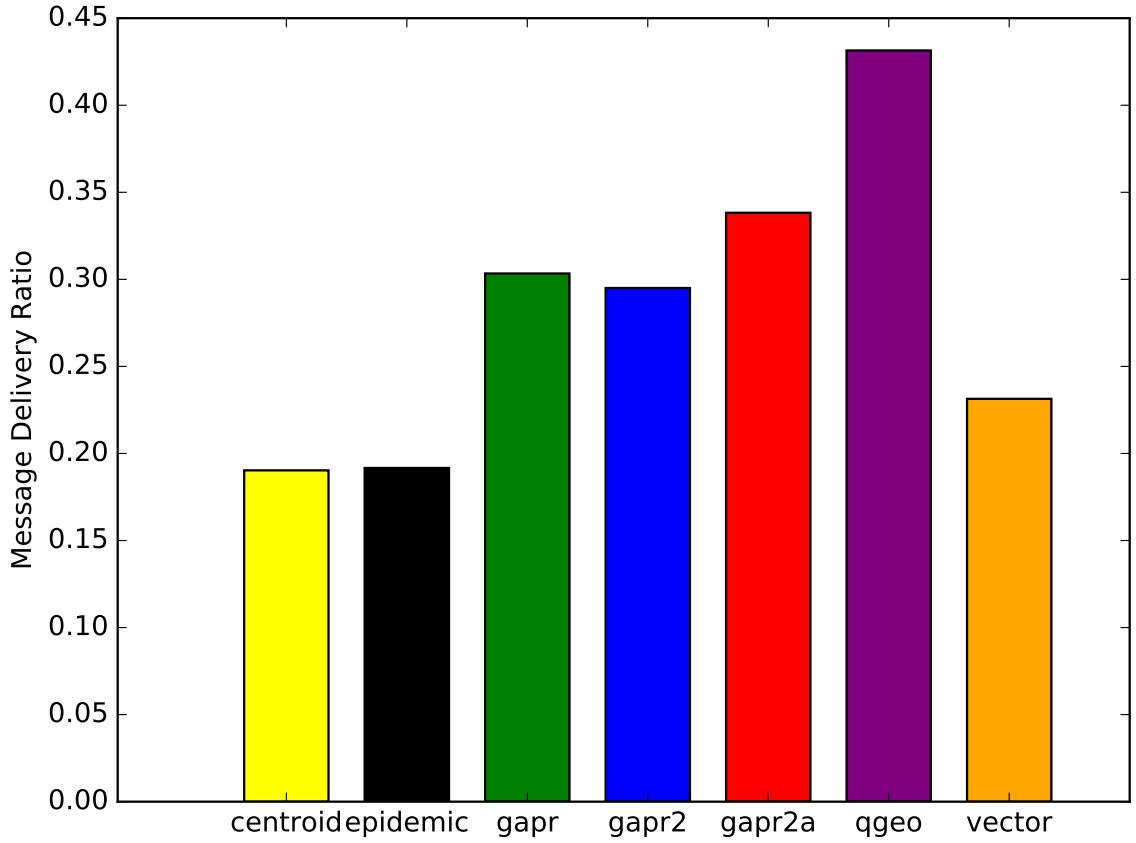


Figure 4.6. Omaha MDR

based on transitive probability, respectively) are forwarded at each interaction. Due to these mechanisms, QGeo and the GAPR protocols are able to better perform in terms of MDR in the Omaha scenario than Epidemic, Vector, and Centroid, and Figure 4.6 shows a relatively clear break in their performance.

4.2.2 Overhead

Figure 4.7 shows the measured overhead values for the Omaha scenario. Unlike the Helsinki scenario, the protocols do not show a correlation in terms of the message forwarding mechanism in the Omaha scenario. This is likely due to the lower node interaction frequency. It is surprising that the GAPR protocols all have a significantly higher overhead than the other protocols, as they do not flood their messages; however, they are still multi-copy routing protocols, so it is likely that the lower number of nodes in the Omaha scenario leads

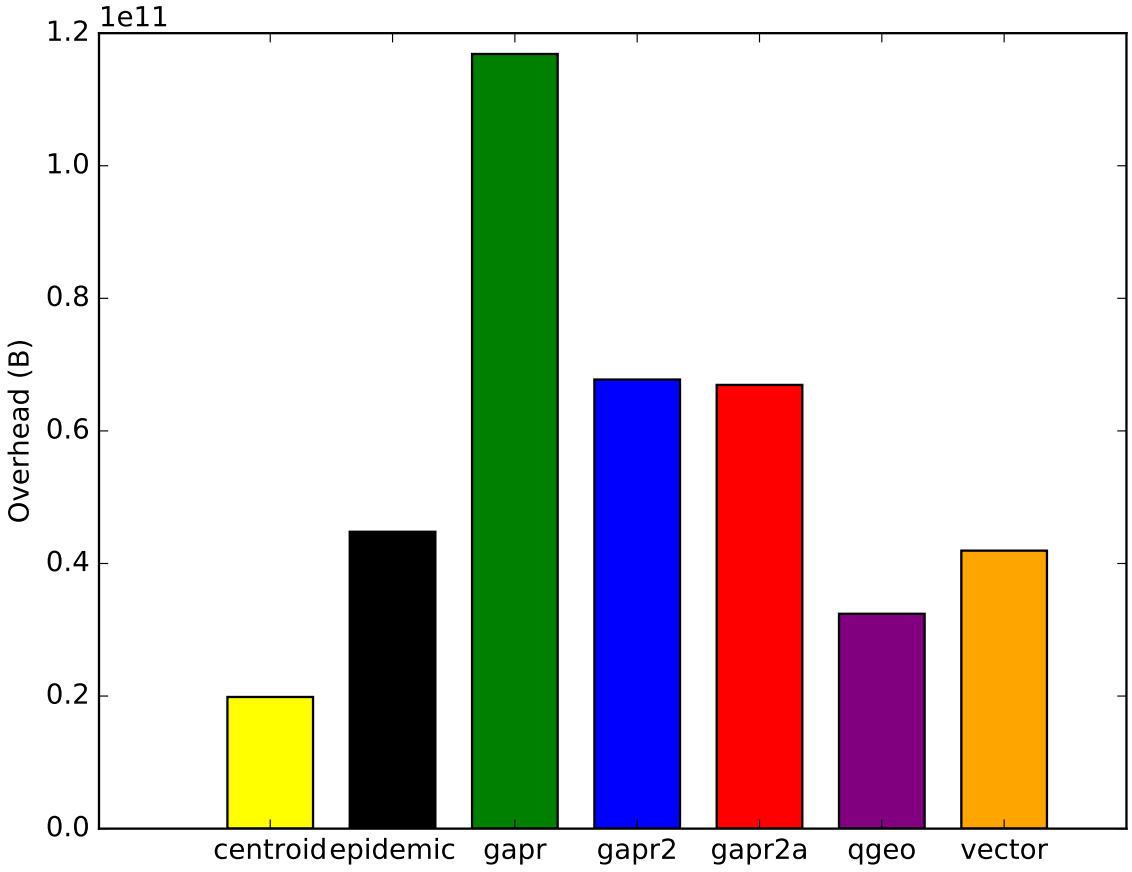


Figure 4.7. Omaha overhead

to the GAPR protocols replicating forwarded messages more frequently than in the Helsinki scenario. Notably, QGeo has one of the lowest overhead values, and when comparing QGeo's relative MDR and relative overhead, both with respect to the other protocols, these results show that QGeo is particularly selective about forwarding messages. The Q-values work to determine that QGeo only forwards messages to the nodes that are most likely to propagate each message to its destination, and limits the likelihood of replicating that message to nodes that have low Q-values.

4.2.3 Goodput

Figure 4.8 shows the goodput results for the Omaha scenario. As in the Helsinki scenario, goodput was calculated as the message size for each message delivered, divided by the amount of time it traveled from transmission to arrival at its destination node, and then

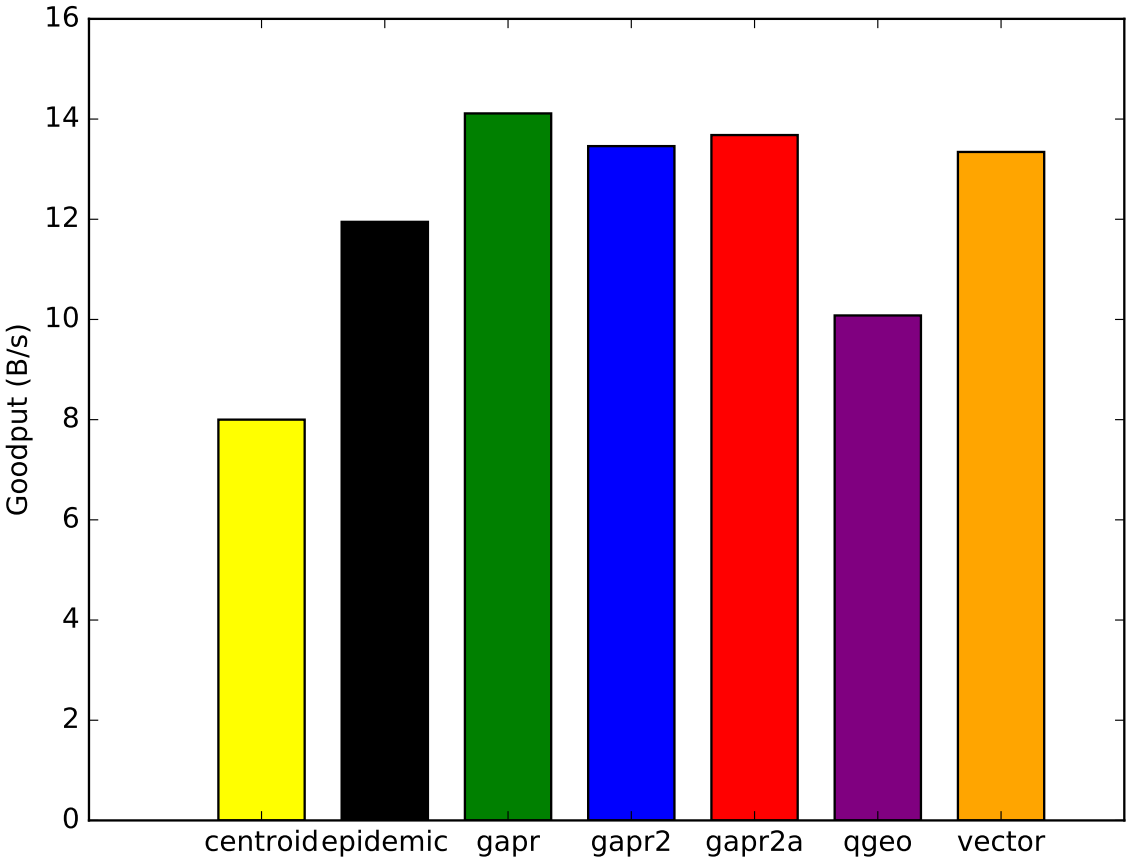


Figure 4.8. Omaha goodput

averaged for all unique messages delivered in one simulation. Those averages were then recorded and averaged to provide the aveage goodput value displayed in Table 4.2 and Figure 4.8. The GAPR protocols and Vector have the same relative performance in Omaha as in Helsinki, but Epidemic, Centroid, and QGeo do not. QGeo's worse relative performance is consistent with expectations based on the scenario differences and its performance measured in the other metrics. Higher MDR, lower overhead and lower goodput all correspond to QGeo holding onto messages longer than the other protocols due to the fewer node interactions and QGeo's use of learned Q-values for message forwarding decisions.

4.2.4 Latency

Figure 4.9 shows the latency results for the Omaha scenario. The distribution of latency values in the Omaha scenario is almost identical to that in the Helsinki scenario, with

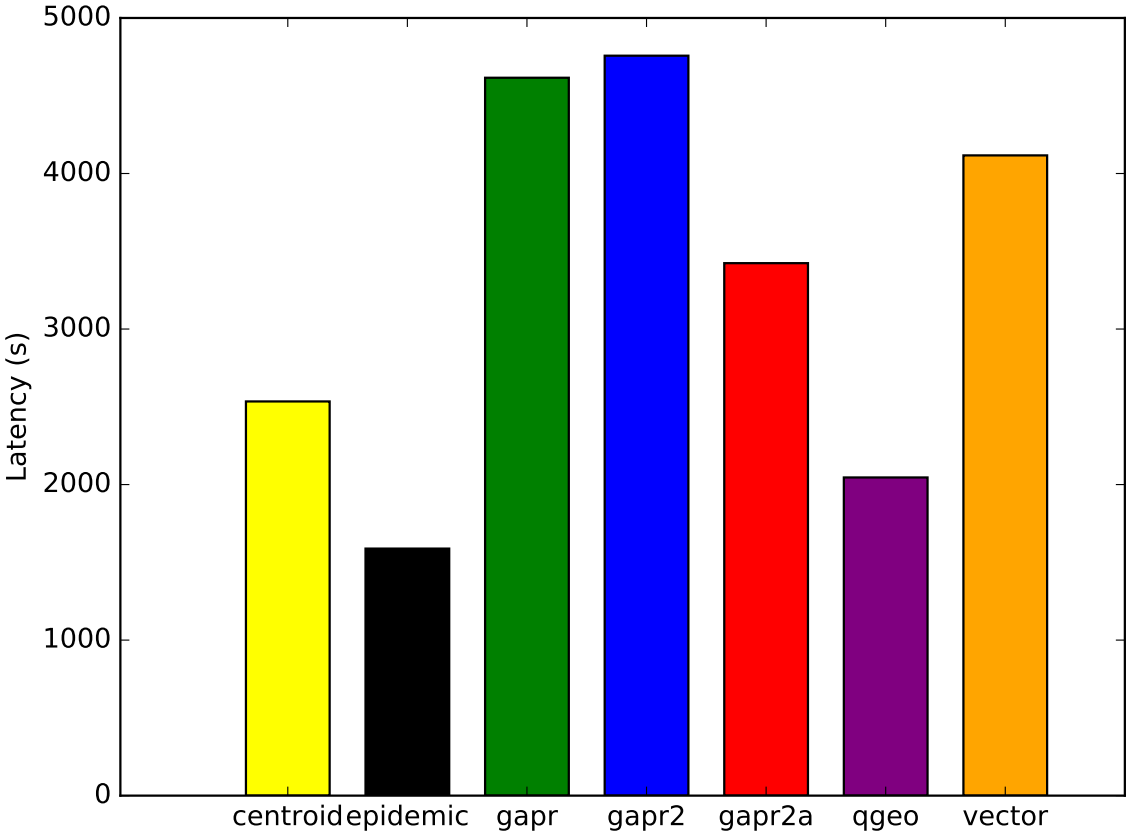


Figure 4.9. Omaha latency

Epidemic having the lowest latency, likely due to its flooding mechanism, and the GAPR protocols having the highest likely due to their queue-based decision making schemes for forwarding messages. As in the Helsinki scenario, QGeo has a low latency, which is likely due to its forwarding and replication decision-making being based on learned Q-values on a per-message basis, rather than based on the queue order overall. This mechanism proves effective to manage overhead, and forward messages in a highly opportunistic manner, which ultimately reduces overall average latency. However, viewing the max latency values described in Table 4.2 reveals that DTNs tend to be highly latent protocols, as is expected considering their design [1]. Since this is the case both for the Omaha and Helsinki scenarios, it is not likely that the average latency is increased solely due to the lower number of nodes in the Omaha scenario.

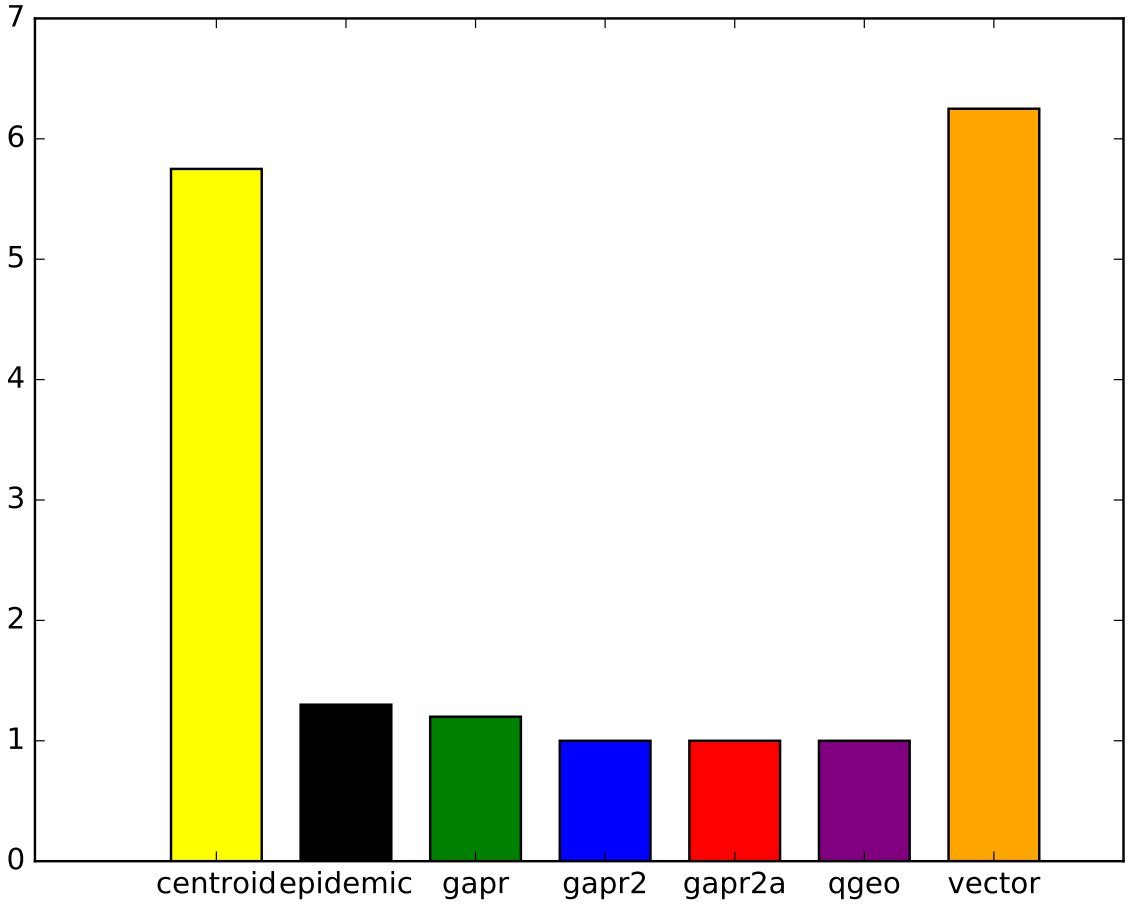


Figure 4.10. Omaha hop count

4.2.5 Hop Count

Figure 4.10 shows the hop count results for the Omaha scenario. As in the Helsinki scenario, the hop count representation for Epidemic, Vector, and Centroid is skewed by the averages being approximated rather than calculated for each simulation and then averaged. The reader is again referred to Mauldin's findings [4] for a more accurate description of the three protocols' performance in the Omaha scenario. As with Helsinki, despite the approximation of Epidemic, Vector, and Centroid, QGeo and the GAPR protocols show greatly reduced hop counts, which are consistent with their message forwarding prioritization schemes (Q-values for QGeo and transitive probability of interaction for GAPR).

4.3 Bold Alligator Results

The Bold Alligator scenario results differ in presentation from the Helsinki and Omaha results, because the Bold Alligator scenario tested the Extended Rate Physical Orthogonal Frequency Division Multiple Access (ERP-OFDM) radio setting in addition to the typical scenario parameters. This study chose Bold Alligator for testing this parameter because the Bold Alligator scenario has the most variation in node radio types, as opposed to the Omaha scenario which only employed one radio type, and Helsinki which only employed two. Bold Alligator employs four heterogeneous radio types, and varying the base radio bandwidth provided data for each protocol's performance over varying radio capabilities while interacting with multiple other nodes that had different radio bandwidth parameters. Subsection 3.3.3 discusses the scenario details. Like the aggregate data displayed in Tables 4.1 and 4.2, Table 4.3 represents the aggregate average statistics for each protocol over all Bold Alligator runs. This representation condenses significantly more data as each Bold Alligator trace file was on average one order of magnitude larger than the Omaha or Helsinki files, and there were four times as many total simulation runs. In its entirety, the Bold Alligator simulation produced almost 6TB of trace file data, while the Omaha, Helsinki and Swarm scenarios produced a total of around 0.75TB of trace data combined. For all data displayed in the Bold Alligator results, the reader should be aware that the GAPR routing protocols had errors when running over the 36Mbps ERP-OFDM rate due to an implementation error in this study, so no data was collected for that parameter on those protocols. All other data in the Bold Alligator scenario is complete. Table 4.3 represents the scenario data in the same manner as all other scenarios.

Table 4.3. Bold Alligator results

Parameter	Epidemic	Vector	Centroid	GAPR	GAPR2	GAPR2a	QGeo
Average Unique Messages Transmitted (K)	13.87	9.18	9.94	8.76	8.81	8.81	4.75
Average Unique Messages Delivered (K)	1.01	1.47	1.46	1.41	1.64	1.5	1.62
Average MDR	0.07	0.16	0.14	0.15	0.19	0.17	0.34
Average Total Transmissions (M)	125,997.97	99,564.06	153,390.79	238,123.28	136,553.76	179,677.23	10,826.93
Average Total Receptions (M)	123,016.26	99,000.53	152,500.33	236,667.26	135,940.53	178,697.03	10,742.21
Average Data Transmitted (GB)	258.04	203.91	314.14	487.68	279.66	367.98	22.17
Average Data Received (GB)	251.94	202.75	312.32	484.69	278.41	365.97	22.0
Average Overhead (GB)	262.81	203.89	314.12	448.12	254.22	352.3	22.16
Average Goodput (B/s)	27.31	39.8	39.51	38.08	44.38	40.68	44.02
Average Latency Mean (s)	1,266.34	3,487.47	3,895.97	5,916.44	5,957.16	3,657.03	2,120.32
Average Latency Median (s)	966.86	2,195.43	2,734.62	4,857.64	4,814.1	1,873.77	0.0
Average Latency Max (s)	14,777.96	16,748.22	16,443.74	16,707.42	17,702.72	16,942.54	17,364.69
Average Hop Count Mean	2.91	15.13	15.53	1.03	1.0	1.0	1.0
Average Hop Count Median	(2.27)*	(10.32)*	(11.38)*	1.06	1.0	1.0	1.0
Average Max Hop Count	1.03	1.0	1.0	4.41	4.0	4.11	3.95

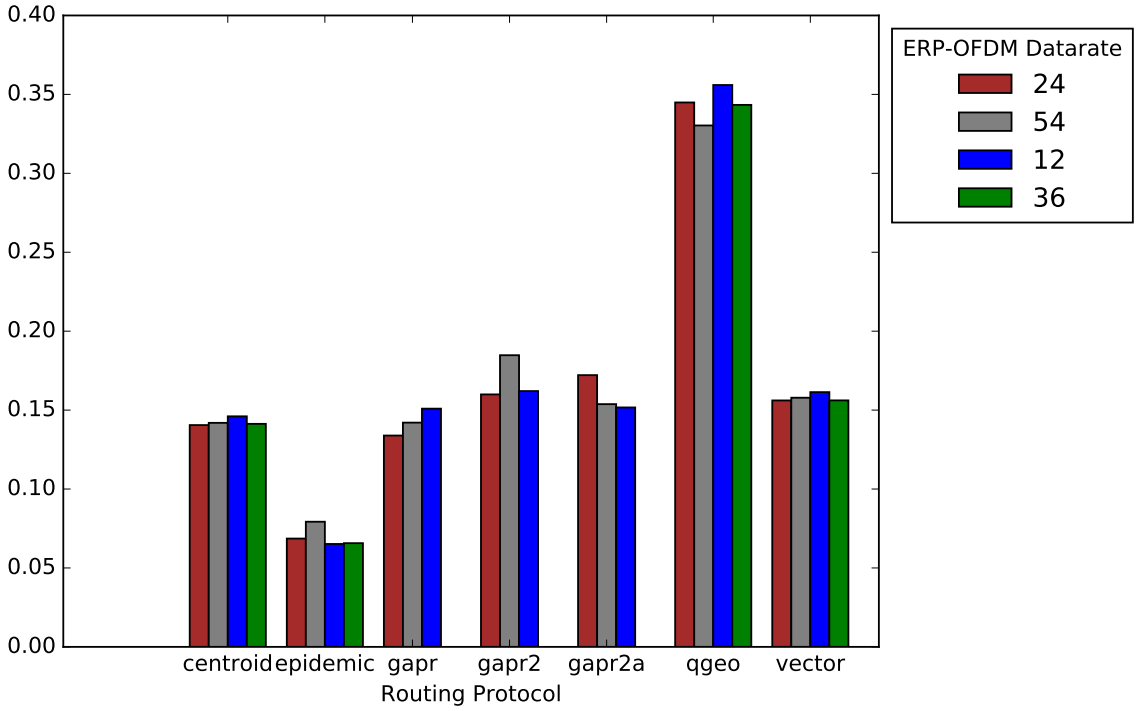


Figure 4.11. Bold Alligator MDR

4.3.1 MDR

Figure 4.11 shows the MDR results for the Bold Alligator scenario. The Bold Alligator MDR results show that each protocol delivers a relatively consistent ratio of messages across all base radio bandwidths. This implies that varying the bandwidth does not increase or decrease the protocols' performance significantly. As with the Omaha scenario, QGeo breaks out above the other protocols in terms of MDR, while Epidemic has a significantly lower average MDR than the other protocols. Similar to Helsinki, all protocols other than QGeo are roughly grouped together, with Epidemic performing significantly worse. Overall, the MDR values of all of the protocols are significantly lower for the Bold Alligator scenario than in the other scenarios. It is likely that this is due to node separation, as the Bold Alligator scenario has the largest operating area of all of the simulation scenarios, and the median number of nodes.

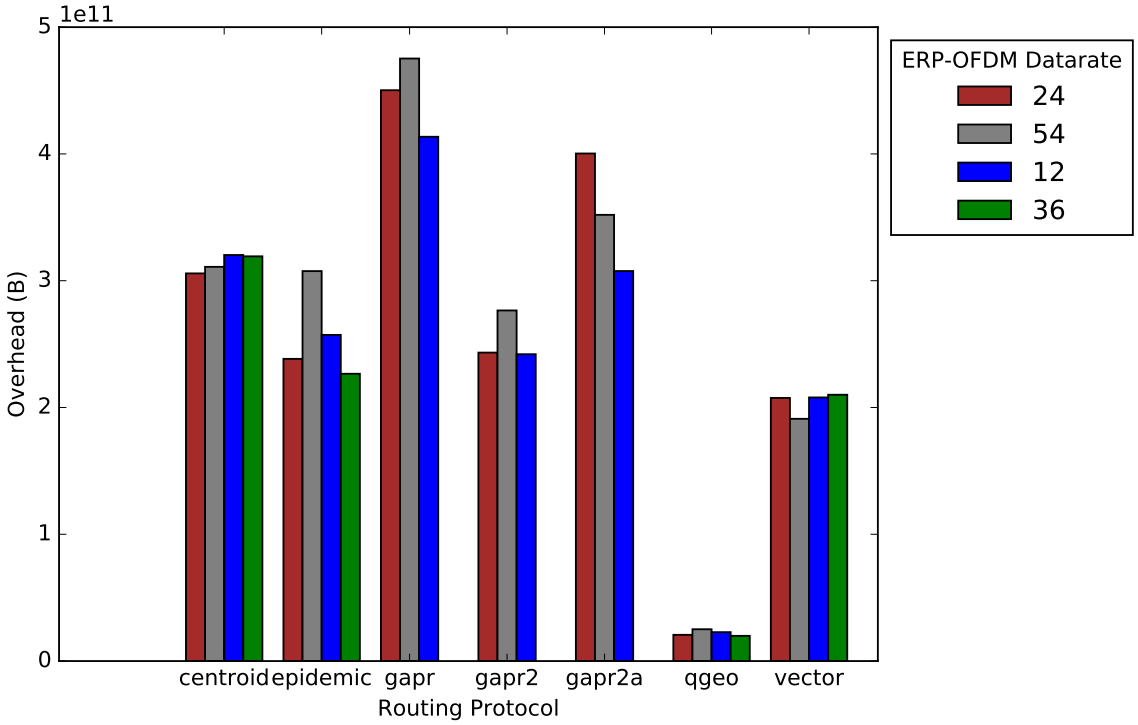


Figure 4.12. Bold Alligator overhead

4.3.2 Overhead

Figure 4.14 represents the Bold Alligator overhead measurements. The Bold Alligator overhead results are most similar to the Helsinki results; however, they exhibit unique characteristics apart from the similarities. Similar to Helsinki, Vector and GAPR2 have similar overhead measurements, and Centroid and GAPR2a are also similar to one another. Alternatively, the most unique feature of the Bold Alligator overhead results is that QGeo severely outperforms the other protocols, maintaining a low overhead around one half of the second-best performing protocol, Vector. QGeo's overhead performance value in the Bold Alligator scenario is very similar to its value in the Helsinki scenario, while all of the other protocols have significantly increased overhead values in the Bold Alligator scenario. As with the Helsinki scenario, these results point to the message forwarding and replication control schemes employed by each of the protocols. In particular, it shows that QGeo's use of learned Q-values significantly aid in message replication and forwarding, while the other protocols' queue management and forwarding limitations do not scale well.

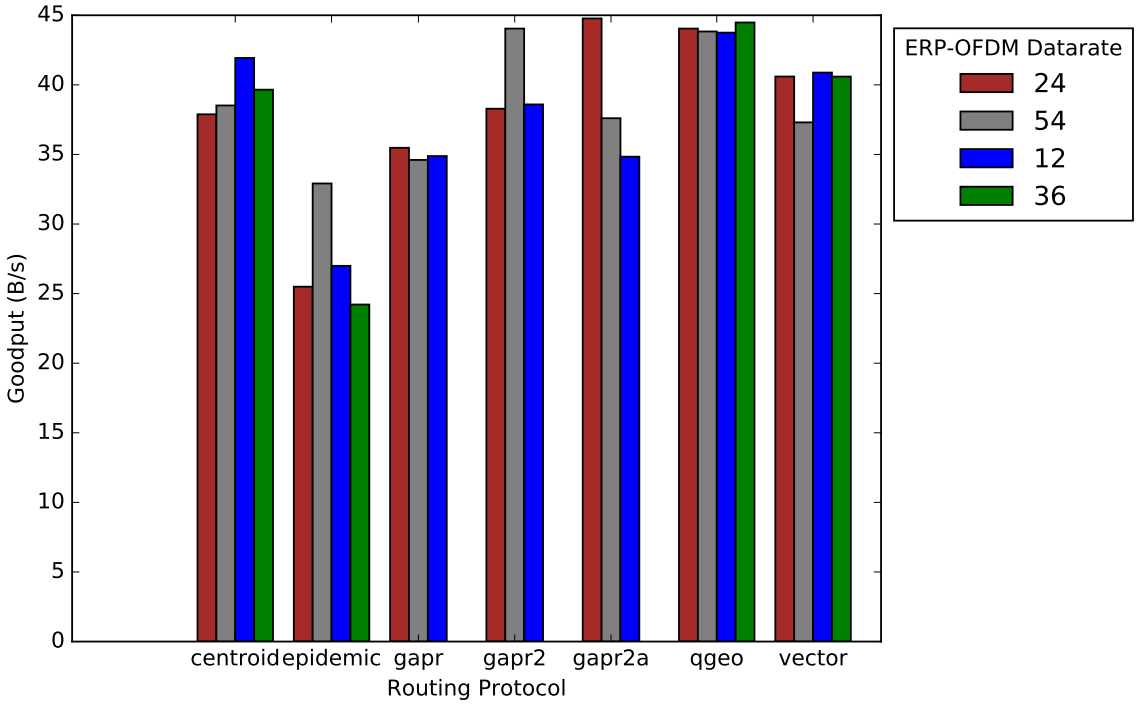


Figure 4.13. Bold Alligator goodput

4.3.3 Goodput

Figure 4.13 depicts the goodput results for the Bold Alligator scenario. The Bold Alligator scenario goodput measurements most closely resemble the Helsinki results as all protocols have goodput values between 25 and 50 B/s, generally. As with overhead, there is no direct correlation or trend across the varied radio bandwidths; however, the same variation per protocol exists in the overhead measurements as in the goodput measurements. In general, Epidemic has the lowest overall average goodput, and QGeo has the most consistent. However, apart from Epidemic, all other protocols perform comparably in terms of goodput.

4.3.4 Latency

Figure 4.14 shows the results for the latency measurements in the Bold Alligator scenario. Similar to the overhead and goodput measurements, latency shows no direct trend across ERP-OFDM rates; however, the relative performance per-protocol for each radio bandwidth are the same at each bandwidth across the overhead, goodput and latency measurements. Broadly speaking, the relative latency performance of the measured protocols matches the

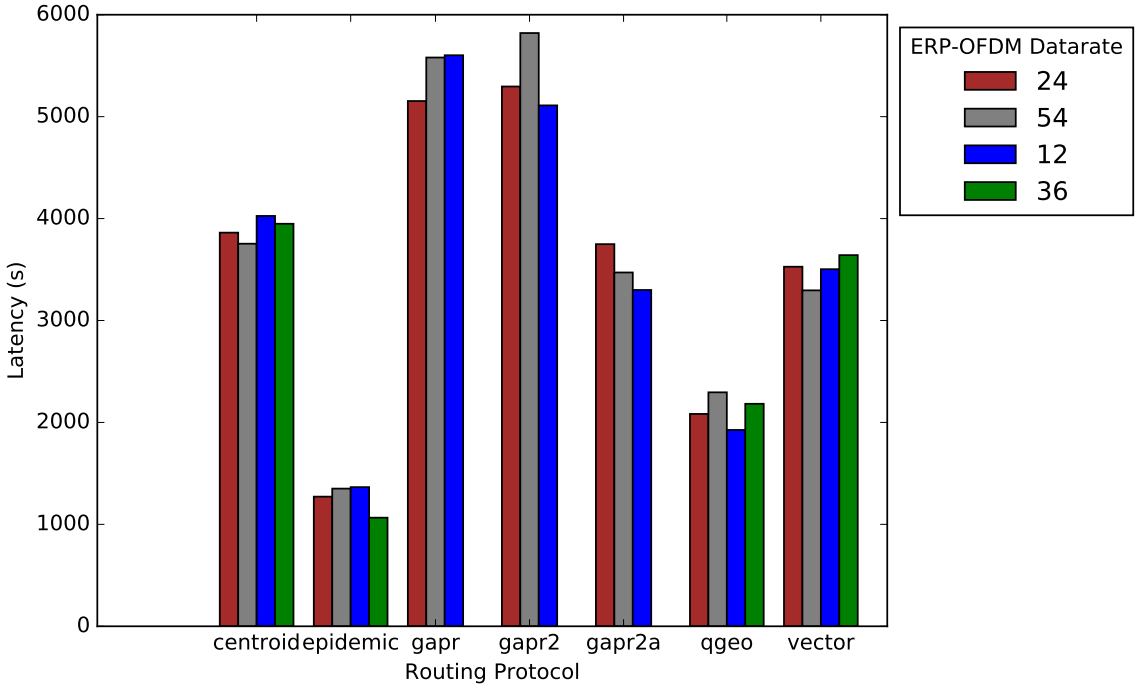


Figure 4.14. Bold Alligator latency

distributions in the Helsinki and Omaha scenario, with Epidemic recording the best (lowest) latencies measured, likely due to its flooding behavior, while GAPR has the worst (highest) latencies measured. QGeo has better latency measurements than all protocols with the exception of Epidemic, which implies that its per-message forwarding decisions, rather than the queue-based scheme used by the other protocols (except for Epidemic), is an effective method for reducing the average latency.

4.3.5 Hop Count

Figure 4.15 depicts the hop count results for the Bold Alligator scenario. As in the Helsinki and Omaha scenarios, the hop count measurements for Epidemic, Vector, and Centroid are all approximate values. QGeo and the GAPR protocols all significantly reduce the total number of hops required to deliver a message, on average. As in the Omaha scenario, Epidemic maintains a low average hop count. Vector and Centroid have significantly higher hop counts than the other protocols. Since Epidemic floods messages, a low hop count is expected, and the GAPR protocols, as well as QGeo use selective forwarding methods where one of the main goals is to reduce the average hop count required for message delivery, so

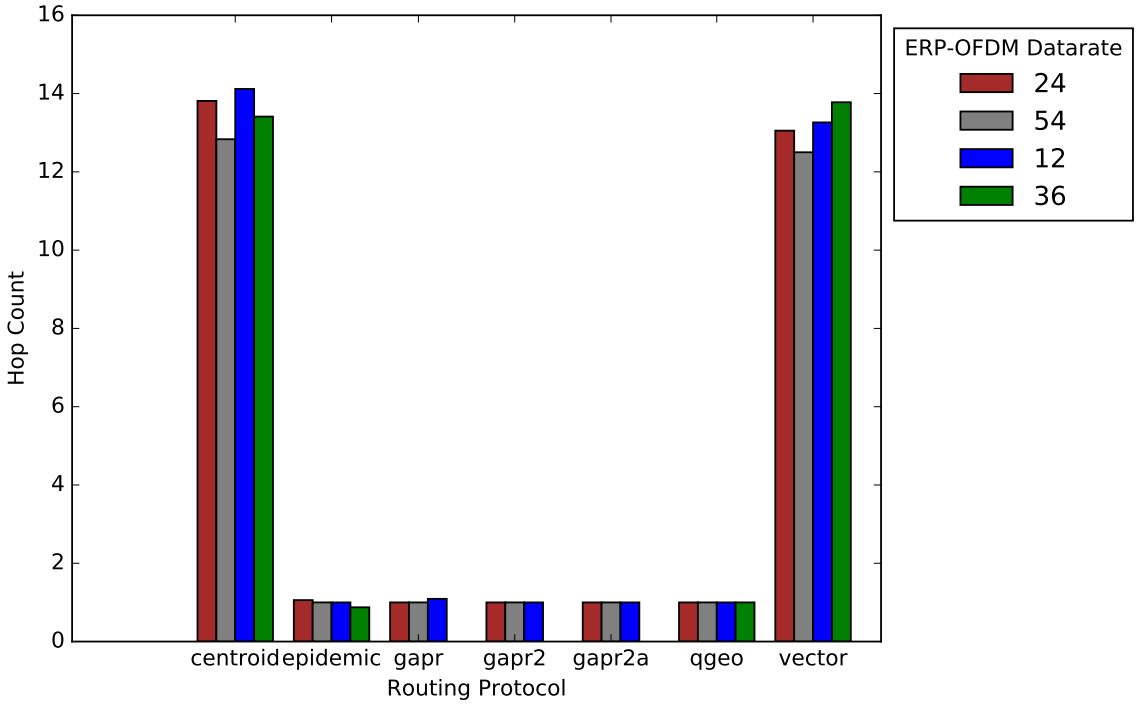


Figure 4.15. Bold Alligator hop count

their behavior is expected as well.

4.4 Swarm Results

The Swarm scenario is significantly different than the other three scenarios. A brief review of Section 3.3 shows that nodes in the Swarm scenario transmit significantly smaller messages at a much faster rate than the other scenarios. The Swarm scenario is also significantly shorter in total scenario time than the other scenarios, due to the limitations of the captured real-world data. Additionally, the nodes maintain the same 5MB buffer limit, so the swarm scenario does not test the buffer management mechanism of the protocols. Instead, the Swarm scenario primarily tests the forwarding and replication decision mechanisms of each of the protocols. Since the Swarm scenario is based on real-world sensor-captured data and real-world radio configurations, the scenario approximates real-world behavior more closely than the other scenarios. In addition to the real-world node behavior, as well as testing the forwarding mechanisms, the Swarm scenario employs a unique parameter among the tested simulations in this study called gain. Gain represents the amplification of the

radio capability, or the attenuation, of the radio. This was first employed in Pospischil's work [5] as a means to enforce multi-hop behavior, because the nodes in the real-world scenario were all one hop away from the destination for every message sent. In other words, every node was always within one-hop communication range of all other nodes. Finally, the Swarm scenario is also unique in that it uses node density as a parameter. Section 3.3 describes the scenario parameters; however, these changes from the Helsinki, Omaha and Bold Alligator scenarios make the Swarm scenario unique, so one of the minor hypotheses of this study was that the protocols would have significant performance differences for this scenario. Table 4.4 illustrates the total aggregate Swarm scenario average data. The table illustrates the aggregate data in the same manner that the other scenarios represent their aggregate data, for comparison purposes, with the exception that the total data transmitted and received values are recorded in megabytes, rather than in gigabytes due to the small message sizes in the scenario. The tables in the subsections that follow are broken into unique plots for each node density parameter, as illustrated by the title "NvN" where "N" is half of the total nodes in the simulation and "v" stands for "versus."

Table 4.4. Swarm results

Parameter	Epidemic	Vector	Centroid	GAPR	GAPR2	GAPR2a	QGeo
Average Unique Messages Transmitted (K)	171.8	139.26	111.66	119.17	120.04	119.5	11.05
Average Unique Messages Delivered (K)	24.38	18.14	13.37	11.37	11.44	11.24	3.44
Average MDR	0.15	0.17	0.12	0.11	0.12	0.11	0.57
Average Total Transmissions (K)	815.89	697.83	529.92	588.93	473.33	526.17	15.49
Average Total Receptions (K)	647.25	567.96	425.08	470.38	361.19	413.51	6.58
Average Data Transmitted (MB)	44.6	38.15	28.97	32.19	25.88	28.76	0.85
Average Data Received (MB)	35.38	31.05	23.24	25.71	19.75	22.61	0.36
Average Overhead (MB)	26.29	19.59	15.69	19.25	12.61	15.16	0.31
Average Goodput (B/s)	866.85	644.81	475.14	404.17	406.67	399.44	122.22
Average Latency Mean (s)	209.6	157.4	34.17	47.84	27.69	23.55	25.27
Average Latency Median (s)	188.26	121.3	7.01	25.44	7.28	4.14	1.15
Average Latency Max (s)	629.62	576.88	260.09	222.51	223.67	218.67	601.22
Average Hop Count Mean	3.25	1.94	2.06	1.33	1.0	1.0	1.0
Average Hop Count Median	(2.61)*	(1.28)*	(1.19)*	1.67	1.03	1.03	1.0
Average Max Hop Count	(1.0)*	(1.0)*	(1.0)*	4.0	1.94	3.39	1.89

Two caveats need to be disclosed for the reader concerning the data for the Swarm scenario. First, there was an implementation error with the Centroid protocol in the Swarm scenario simulation file. This was realized late in testing, so the Centroid protocols did not fully complete their testing. At the time of publishing, 40% of the simulations for Centroid were totally complete, and the rest of each of the remaining simulation runs were at least halfway complete (complete up to 750s). Second, the GAPR protocols run significantly slower in the Swarm scenario than the other protocols in the scenario, and significantly slower than in other scenarios. No GAPR runs completed running, and were cut off after roughly one

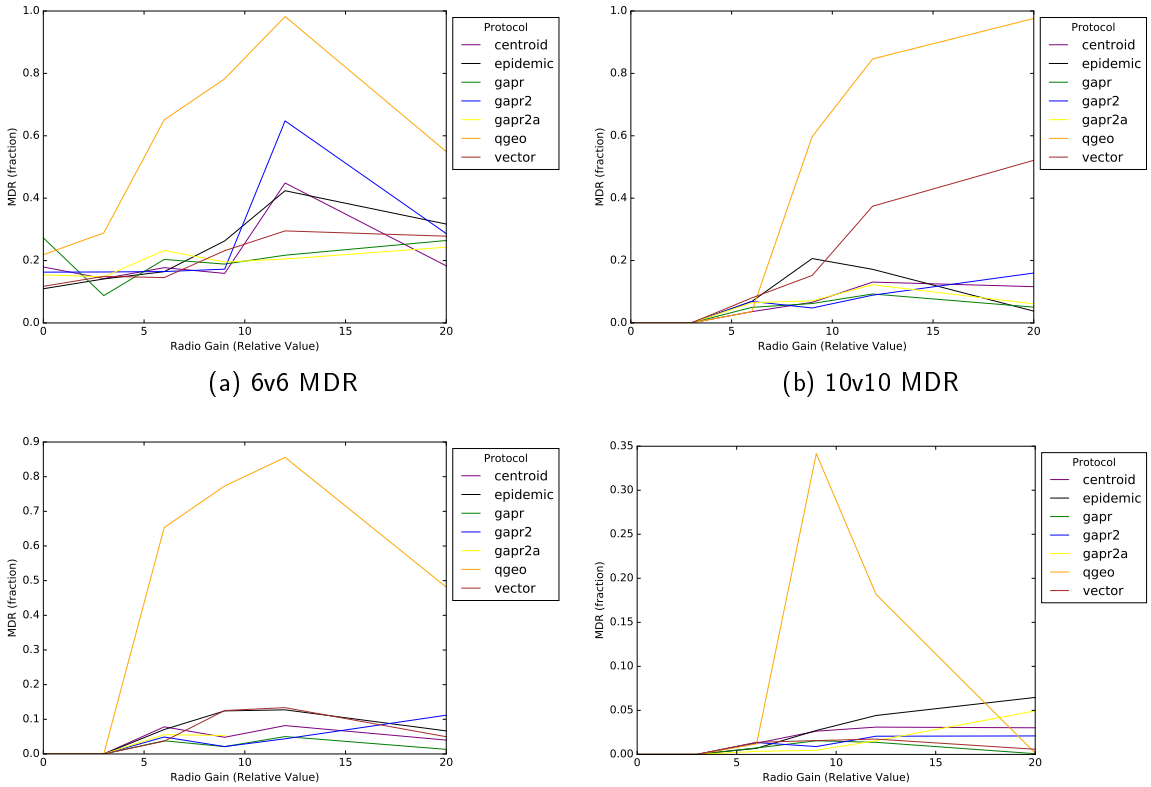
month of running, for each simulation run. In one month of real-world running time, on average, each file made it roughly half way. It was observed that Swarm simulation runs with the GAPR protocols installed would get to a point between 600s and 800s in simulation time in about one day in real world time. After that point, each simulation ceased to progress at a reasonable rate, and only made another 100-200s progress in the following month. After significant testing, this study believes that the cause for this issue lies in the rate and total number of messages that the Swarm scenario generates, combined with the GAPR protocols' sorting behavior. None of the other protocols sort their buffer, but the GAPR protocols sort their buffer at every node interaction, as a means for determining which messages will be forwarded to the interacting node. Since the Swarm scenario generates multiple messages at each node every second, it is likely that the nodes add messages to their buffers so quickly that the total number of messages to sort is intractable for the protocol to handle in a reasonable timeframe. Finally, GAPR2a runs failed for the 12 and 20 gain tests, while all other simulations utilizing the GAPR protocols were simply unable to complete in a reasonable time. Further testing of the Swarm scenario and/or the GAPR protocols should focus on this challenge.

4.4.1 MDR

Figure 4.16 represents the average Swarm MDR measurements for each node density. The Swarm MDR measurements, as with all of the other metrics, were measured over multiple gain values, represented on the x-axis of the plots. The line plots show how the protocols either increase or decrease in MDR performance with changing gain values. The trends in each of the unique node density plots are generally similar to one another with QGeo outperforming the other protocols at higher gains, and the rest of the protocols generally clustering as node density and gain both increase. QGeo's performance in terms of MDR is consistent with the other scenarios, indicating that in general, QGeo forwards messages most opportunistically, when compared with the other tested protocols.

4.4.2 Overhead

Figure 4.17 depicts the average overhead values of the protocols measured in the Swarm scenario. The reader should note that the Swarm overhead plots are depicted in log scale, whereas the Helsinki, Omaha, and Bold Alligator overhead plots are in linear scale. Gener-



(a) 6v6 MDR

(b) 10v10 MDR

(c) 15v15 MDR

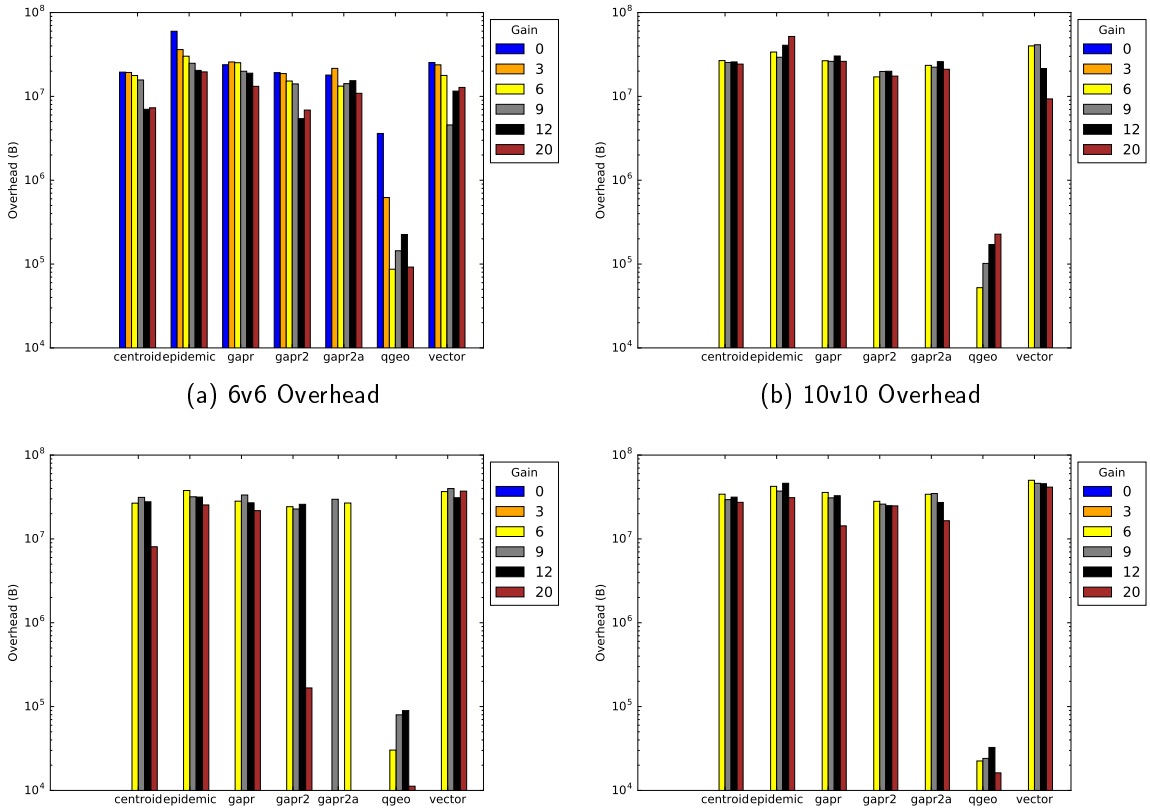
(d) 25v25 MDR

Figure 4.16. Swarm MDR

ally speaking, the protocols tend to have very slightly decreasing overhead as gain increases, with the exception of QGeo. QGeo has the lowest overhead values of all the protocols, with averages generally two orders of magnitude lower than the other protocols. This is indicative of QGeo's selective behavior in terms of message forwarding and replication. Similar to the other scenarios, QGeo exhibits a strong limitation on forwarding messages except in node interactions that are likely to deliver the message to its destination. Notably, the overhead data shows that past the 12-node simulation, no protocols are able to successfully transmit data for the lower gains (below 6).

4.4.3 Goodput

Figure 4.18 represents the average goodput values measured for each of the protocols in the Swarm scenario. As with the overhead measurements, the goodput values show



(a) 6v6 Overhead

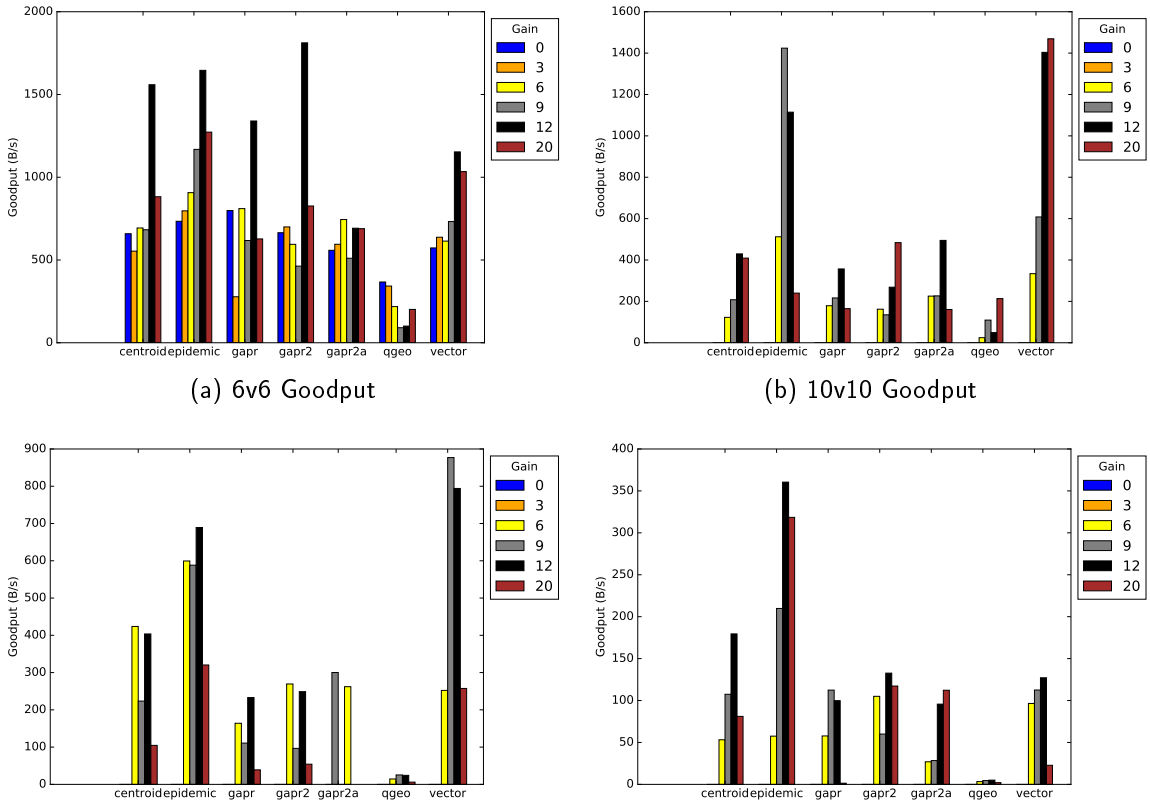
(b) 10v10 Overhead

(c) 15v15 Overhead

(d) 25v25 Overhead

Figure 4.17. Swarm overhead

an increasing trend as gain increases, except for QGeo, which seems to have an inverse relationship with increasing gain. Additionally, variations in the Centroid and GAPR protocols' trends can be attributed to issues with their simulation runs. Beyond the trends for each of the protocols, QGeo has a distinctly lower goodput value for all gains in all node densities than all of the other protocols. As node density increases, Epidemic tends to break out above the other protocols in terms of average goodput. These goodput values make sense when compared to the MDR results and total data transmitted and delivered as described by Table 4.4. QGeo has a high MDR, but very little overall data delivered when compared to the other protocols. This corresponds to the low average goodput shown in Figure 4.18. Alternatively, Epidemic has an improving MDR as node density increases, and has the highest average data delivered value of all of the protocols. This corresponds to Epidemic's increasingly dominant performance in the goodput results.



(a) 6v6 Goodput

(b) 10v10 Goodput

(c) 15v15 Goodput

(d) 25v25 Goodput

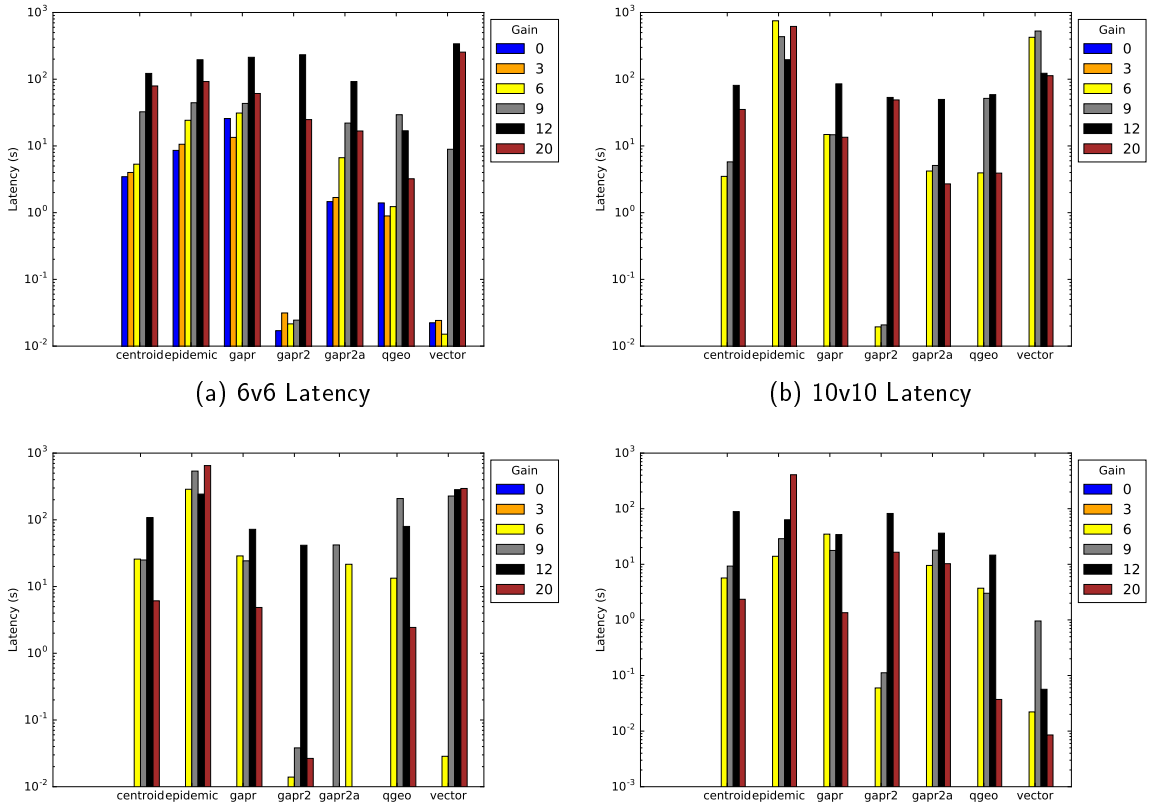
Figure 4.18. Swarm goodput

4.4.4 Latency

Figure 4.19 shows the average latency values measured for the Swarm scenario. The 12-node simulations show a clear trend that as gain increases, so does latency. The other simulations do not follow the same trend, and in general, all of the protocols tend to perform generally within an order of magnitude of one another in terms of message delivery latency. Vector seems to perform slightly better in the 50-node scenario than the other protocols, but the other node density scenarios do not show the same trend, so the reason for this performance is unclear.

4.4.5 Hop Count

Figure 4.19 depicts the average hop count values for the Swarm scenario. As with all of the other scenarios, Epidemic, Vector, and Centroid hop count values are approximated



(a) 6v6 Latency

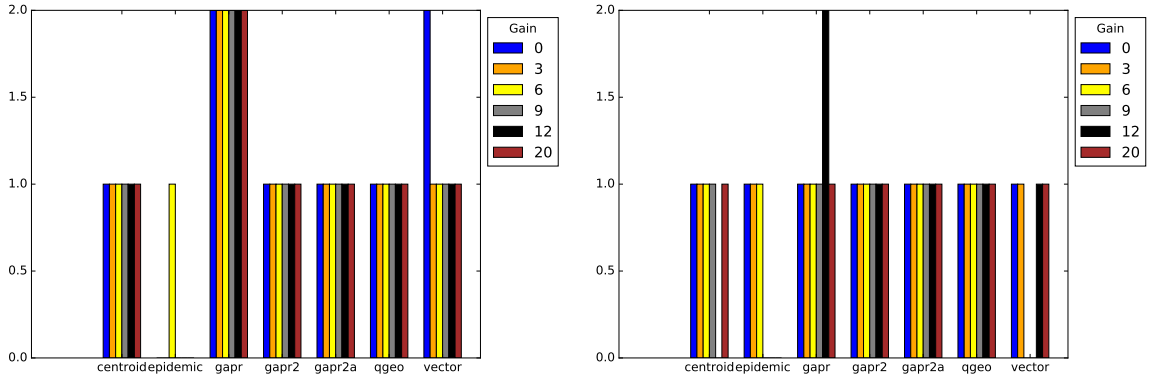
(b) 10v10 Latency

(c) 15v15 Latency

(d) 25v25 Latency

Figure 4.19. Swarm latency

due to implementation issues. However, even with approximated values and comparing the plots to Table 4.4, the Swarm scenario exhibits very low average hop count values. This is likely due to the real-world behavior of the swarm. Since all nodes were one hop from their destination, it is likely that the gain values employed did not significantly impact the routing protocols ability to transmit messages from the aerial nodes to the ground station. Ultimately, this indicates that at various times, each node was able to pass close enough to the ground station to transmit messages without relying on multi-hop forwarding. This result indicates that further work should encompass other gain values to potentially enforce multi-hop behavior in a more effective manner for DTN protocols.



(a) 6v6 Hop Count

(b) 10v10 Hop Count

(c) 15v15 Hop Count

(d) 25v25 Hop Count

Figure 4.20. Swarm hop count

CHAPTER 5:
Conclusions and Future Work

This Chapter discussions conclusions taken from the analysis portion of this study, as well as potential follow-on future work. No data is presented in this chapter; however, all conclusions are drawn from the data presented in Chapter 4 and the general findings from this study.

This thesis conducted rigorous testing for a machine-learning-based routing protocol, namely QGeo. Additionally, this thesis implemented improvements to the Swarm scenario to increase its realism. Section 5.1 presents conclusions for QGeo's performance compared to the other protocols, specific notable attributes of the other protocols, and the benefits of the testing framework employed in this study. Future work is presented in Section 5.2 correllating to the two major efforts of this study: the Swarm scenario and machine-learning-based network routing protocols.

5.1 Conclusions

In general, QGeo did not stand out in performance in the Helsinki and Omaha scenarios. QGeo's performance in the Helsinki and Omaha scenarios was generally most similar to the GAPR protocols; however, in the Bold Alligator scenario, QGeo had more desirable overall attributes than all of the other protocols. It had a higher MDR and comparable goodput, with a lower average overhead than all other protocols and lower latency than all other protocols except Epidemic. In the Swarm scenario on the other hand, QGeo had better MDR and overhead averages for all gains, but lower goodput and comparable average latency and hop count values. In general, QGeo's performance indicates that it is better than other protocols in scenarios with low node density where interactions are infrequent and messages need to be held until opportune interactions for forwarding to their destinations. Another general finding for QGeo is that it seems to exhibit direct contact behavior. In other words, while QGeo does not have a constant hop count rate of one, meaning it does forward messages, the low hop counts observed across all scenarios indicate QGeo is likely sending messages directly to their destination, more often than not. A possible cause of this behavior is that the learning mechanism is prioritizing the destination Q-Values for each message to the

extent that it is inducing the direct contact behavior. It is possible that the learning rate is not learning best forwarding nodes, but that the highest Q-Value belongs to the destination, and destination contact is frequent enough to allow nodes to transmit messages directly to their destinations.

It is hard to compare QGeo's performance to Jung, Yim and Ko's original introduction [2], as their original implementation is unavailable to this study and the swarm scenario they employed was not tested in this study. This study attempted to validate the tested implementation of QGeo by ensuring the reported Q-Values were within expected limitations. This means that during the implementation phase, the running Q-Values were printed out with their corresponding message destination, and the node from which they were being printed. Once these values were within the normalized range [-1, 1], this study assumed the Q-Value update mechanism worked, as there were no expected values described in [2]. For this reason, there is potential room for the QGeo implementation tested in this study to be refined: further testing could be done to validate that the implemented version of QGeo in this study produces the same or similar performance using the context described in [2]. Nonetheless, since QGeo was designed to be useful for drone swarms, its high relative MDR and overhead performance is encouraging, while its low goodput performance leaves room for further improvements.

The main conclusion from this study concerning QGeo is that its message forwarding determination, based on Q-values on a per-message basis, is an efficient method that results in highly selective forwarding behavior. The learning mechanism that employs the determined Q-values seems to greatly decrease protocol overhead without significantly reducing average goodput when compared to the other protocols tested. The other significant result from this study concerning QGeo indicate that QGeo's buffer management is better than the GAPR protocols' scheme because it does not perform any buffer sorting. Alternatively, QGeo is worse than Epidemic, Vector, and Centroid in terms of buffer management because it performs a calculation for every message at each interaction to determine the updated Q-value for that message, rather than simply forwarding messages based on a FIFO scheme. This illustrates the tradeoff nature of complex protocols. Where improvement is seen in one area, there is generally a decrease in performance in another area. Overall, it is the belief of this study that the tested QGeo implementation provides a good platform for designing and testing machine-learning-based protocols in a simulation environment.

This study provides further rigor to previous testing in Mauldin's [4] and Killeen's [3] work in validating the GAPR protocols' improved performance over Epidemic, Vector, and Centroid in terms of MDR and controlling message replication overhead. While GAPR has a consistently higher overhead, GAPR2 and GAPR2a significantly reduce the overhead below all of the other protocols. Since they use the message forwarding limitation mechanisms of Vector and Centroid, respectively, it makes sense that they reduce overhead significantly, because the forwarding limitation schemes are designed to reduce the total number of messages transmitted. Additionally, Epidemic generally has low latency and low hop count values across all simulation scenarios, despite its high overhead. This makes sense because its flooding mechanism replicates messages at every interaction in order to quickly and effectively get messages to their destinations, which is what MDR and latency measure, respectively. Vector and Centroid do not have notable performance in any of the metrics in general, but serve as useful comparisons in particular for GAPR2 and GAPR2a.

Finally, one of the major contributions of this study was to add an improved Swarm scenario to the testbed of simulations used to study DTNs. This ultimately takes the pattern of testing DTNs from Killeen's study [3], furthered in Mauldin's study [4], and continues to push it forward with an extension of Pospischil's study [5]. This testbed provides rigorous testing through the use of various node densities, several mobility models including many node types and velocities, and varying radio parameters and types. These variations are highly useful for testing a wide range of potential applications for implemented routing methodologies. Beyond routing, ns3 allows for various network characteristic modifications such as propagation delay and jamming that can be used in these scenarios to test other network performance characteristics not even mentioned in this study.

5.2 Recommended Future Work

There are many improvements that can be made to the work of this study, in order to further the testing of the routing protocols employed, or to allow for further network characteristic testing in general. In particular, in terms of the simulation scenarios, the Swarm scenario could be improved by a refinement of the gain parameter. The current findings seem to suggest that the tested gains may not be taking advantage of the full effect of modifying the radio attenuation for the purpose of inducing multi-hop behavior in a network that does not naturally require multiple hops for message delivery. This is evidenced by the

Swarm hop count results in Subsection 4.4.5. Future work could also incorporate more real-world data and new node types. Collaboration with outside organizations could provide more real-world flight log data that would enable the realistic nature of the tested Swarm scenario to continue, while increasing the sensor data diversity and refining the acquired sensor data by obfuscation of errant data. This will also likely increase the operating range of the swarm, and in turn, decrease node density. Decreased node density is a highly desirable characteristic for the Swarm scenario, as the results in Section 4.4 indicate that the nodes have many interactions and generate messages at too high a rate for more advanced protocols like QGeo and the GAPR protocols to be highly effective. While it is not a negative result that those protocols are less effective in the Swarm scenario, the scenario is from a small set of flight data that is likely not representative of potential drone swarm operations in a real-world contested environment. NPS continues to pursue research in advances in drone swarm technology, and as drones become more prevalent, it will be useful to know how varying radio types can be used in various swarm configurations with specific routing methodologies to the greatest effect. The potential for further research in this field is widespread.

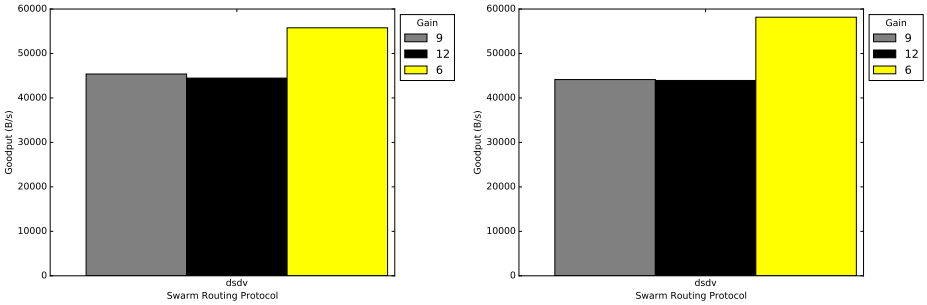
In terms of the usefulness of machine-learning-based protocols, it is the view of this study that in certain scenarios QGeo is more effective than other more common protocols, and performs comparably at worst to an advanced hybrid family of protocols, GAPR. In particular, scenarios that need routing protocols with the characteristics of long message holding times, and opportunistic forwarding would benefit from the use of QGeo. For commonly used reinforcement learning schemes like Q-learning or SARSA, this finding can be extrapolated, but with the advent of many different machine learning schemes, it is impossible to extract the same routing methodology. With that in mind, the easiest follow-on work to this study would be to implement other more modern protocols in ns3 and continue this testing framework to compare how QGeo performs against the most modern DTN routing protocols. Another direct extension of this thesis' work would be to examine total simulation time as a parameter to see if it induces a different behavior from QGeo based on the protocol's learning mechanism. A more original follow-on work could easily implement a SARSA learning-based protocol based on the QGeo implementation from this study, while a more challenging study could attempt to extrapolate the simulation testbed onto a Deep Q network and use a similar Q-learning scheme over multiple layers of

learning to develop a policy-based protocol that would operate optimally for a wide range of scenarios. These opportunities for further work were outside the scope of this study, but are reasonable for an improvement to the work conducted in this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

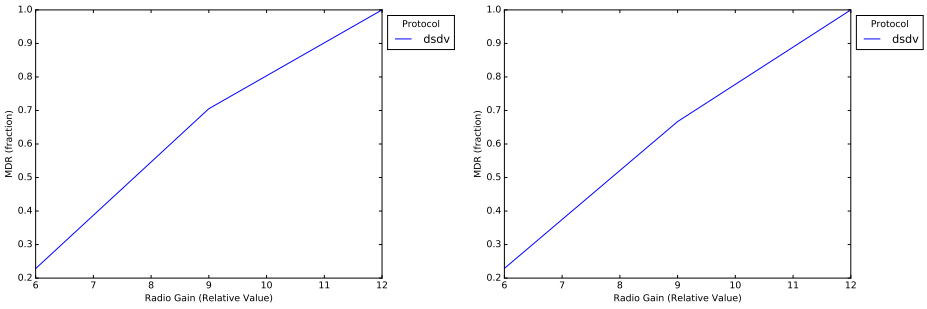
APPENDIX: Swarm Mobility Data

A.1 6v6 Mobility Case Study (DSDV) Results



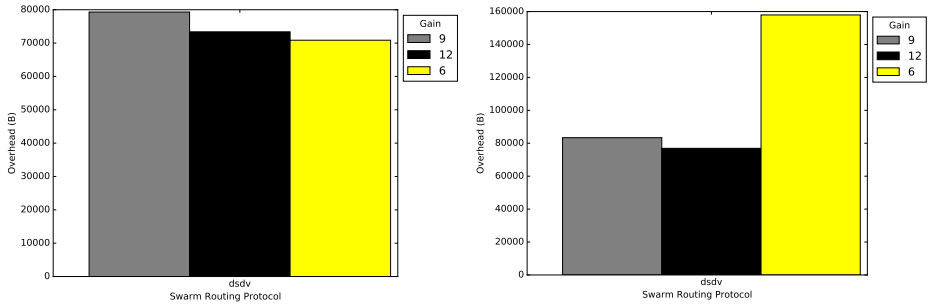
Constant Position Mobility Model ns3 Mobility Model

Figure A.1. 6v6 goodput



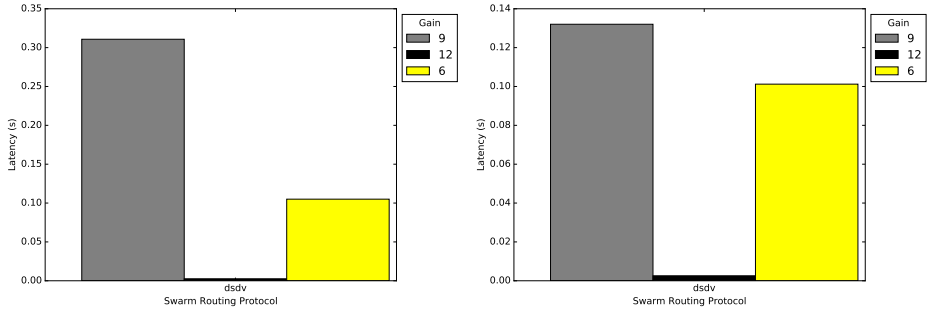
Constant Position Mobility Model ns3 Mobility Model

Figure A.2. 6v6 MDR



Constant Position Mobility Model ns3 Mobility Model

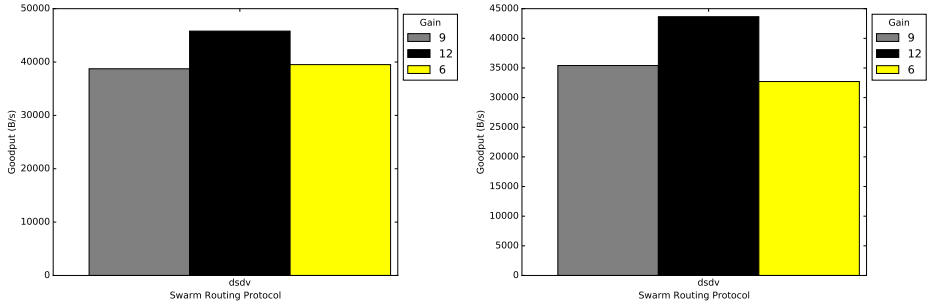
Figure A.3. 6v6 overhead



Constant Position Mobility Model ns3 Mobility Model

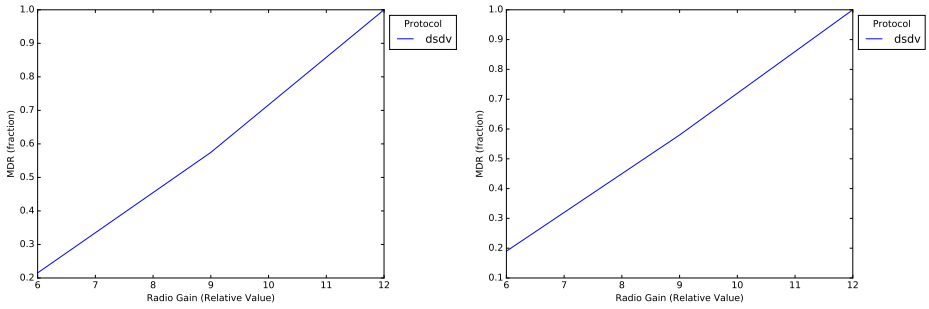
Figure A.4. 6v6 latency

A.2 10v10 Mobility Case Study (DSDV) Results



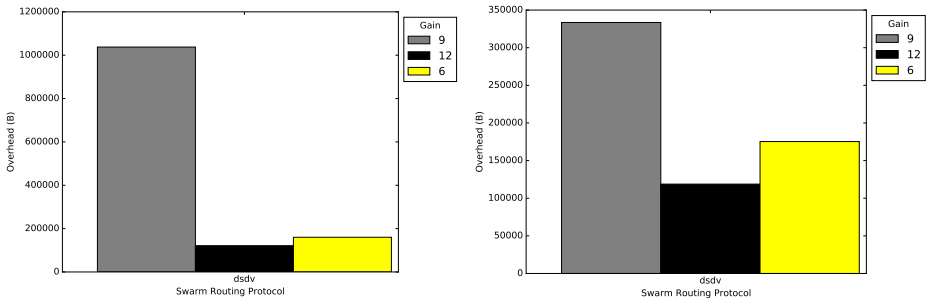
Constant Position Mobility Model ns3 Mobility Model

Figure A.5. 10v10 goodput



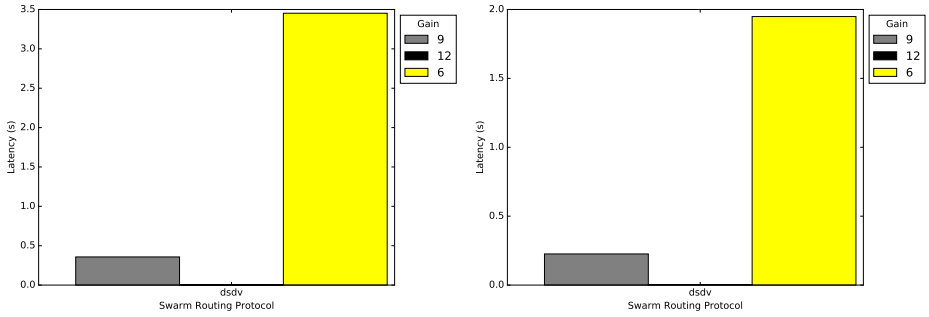
Constant Position Mobility Model ns3 Mobility Model

Figure A.6. 10v10 MDR



Constant Position Mobility Model ns3 Mobility Model

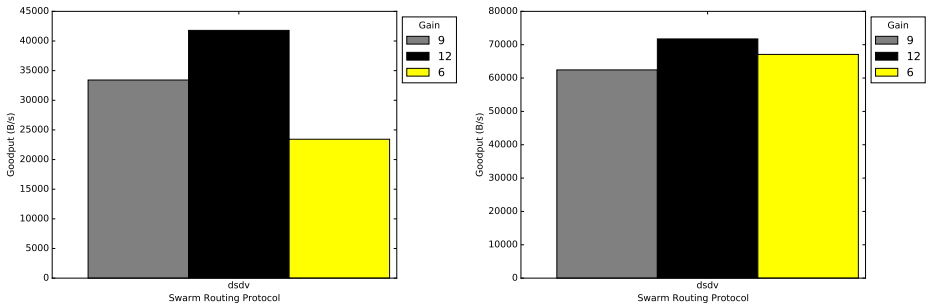
Figure A.7. 10v10 overhead



Constant Position Mobility Model ns3 Mobility Model

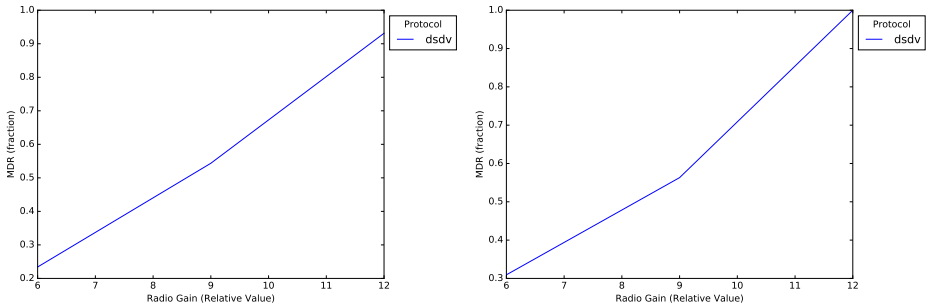
Figure A.8. 10v10 latency

A.3 15v15 Mobility Case Study (DSDV) Results



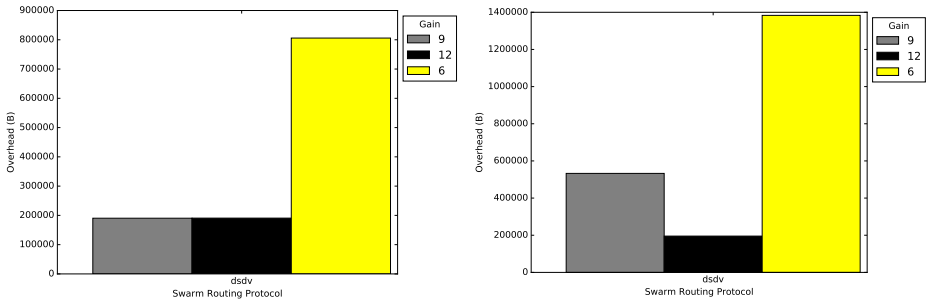
Constant Position Mobility Model ns3 Mobility Model

Figure A.9. 15v15 goodput



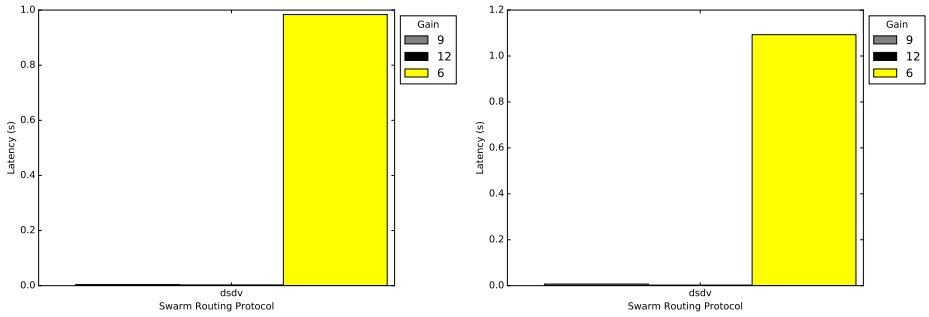
Constant Position Mobility Model ns3 Mobility Model

Figure A.10. 15v15 MDR



Constant Position Mobility Model ns3 Mobility Model

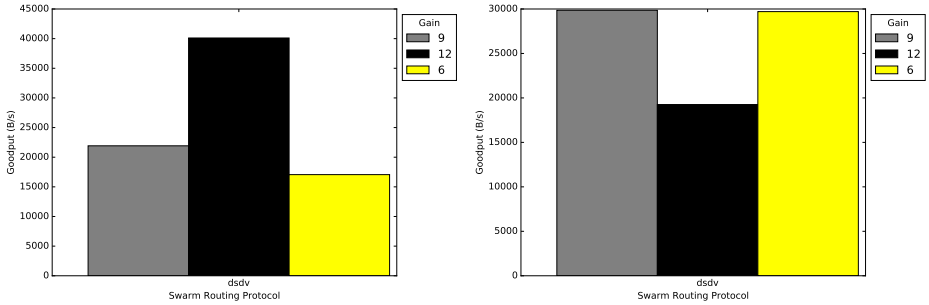
Figure A.11. 15v15 overhead



Constant Position Mobility Model ns3 Mobility Model

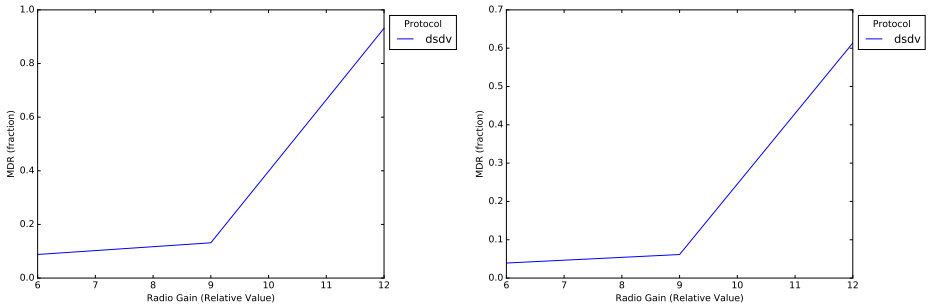
Figure A.12. 15v15 latency

A.4 25v25 Mobility Case Study (DSDV) Results



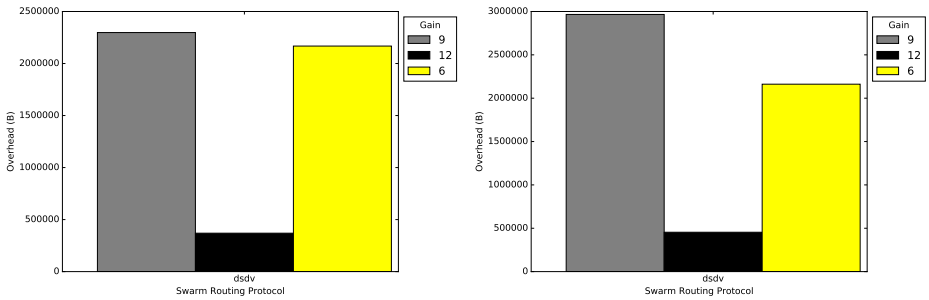
Constant Position Mobility Model ns3 Mobility Model

Figure A.13. 25v25 goodput



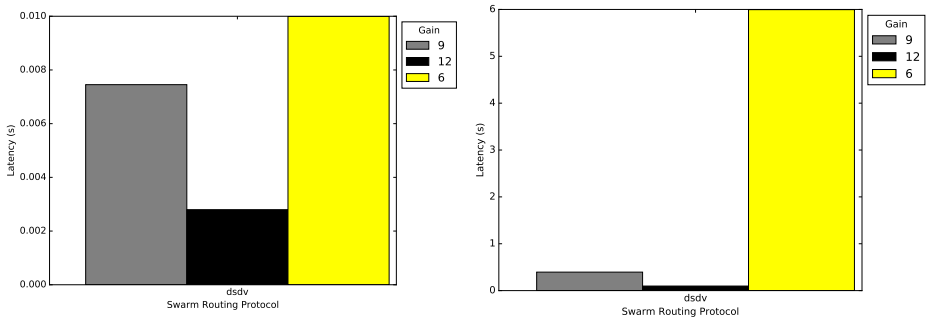
Constant Position Mobility Model ns3 Mobility Model

Figure A.14. 25v25 MDR



Constant Position Mobility Model ns3 Mobility Model

Figure A.15. 25v25 overhead



Constant Position Mobility Model ns3 Mobility Model

Figure A.16. 25v25 latency

List of References

[1] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-tolerant networking architecture," RFC 4838, 2007.

[2] W.-S. Jung, J. Yim, and Y.-B. Ko, "Qgeo: Q-learning-based geographic ad hoc routing protocol for unmanned robotic networks," *IEEE Communications Letters*, vol. 21, no. 10, pp. 2258–2261, 2017.

[3] K. M. Killeen Jr, "Gapr2: A dtn routing protocol for communications in challenged, degraded, and denied environments," M.S. thesis, Dept. Comp. Sci., Naval Postgraduate School, Monterey, CA, United States, 2015.

[4] A. N. Mauldin, "Comparative analysis of disruption tolerant network routing simulations in the one and ns-3," M.S. thesis, Dept. Comp. Sci., Naval Postgraduate School, Monterey, CA, United States, 2017.

[5] A. Pospischil, "Mapping ad hoc communications network of a large number fixed-wing UAV swarm," M.S. thesis, Dept. Comp. Sci., Naval Postgraduate School, Monterey, CA, United States, 2017.

[6] J. Brown and J. P. Rohrer, "DTN routing protocols for drone swarm telemetry," in *Proceedings of the International Telemetering Conference (ITC)*, Phoenix, AZ, November 2018, pp. 1–10.

[7] J. P. G. Sterbenz, J. P. Rohrer, M. J. Alenazi, T. A. N. Nguyen, E. K. Çetinkaya, H. Narra, K. S. Pathapati, and K. Peters, "Disruption-tolerant airborne networks and protocols," in *UAV Networks and Communications*, K. Namuduri, S. Chaumette, J. H. Kim, and J. P. G. Sterbenz, Eds., 1st ed. Cambridge University Press, January 2018, ch. 4, pp. 58–95. Available: https://doi.org/10.1017/9781316335765

[8] J. P. Rohrer, A. Jabbar, E. K. Cetinkaya, E. Perrins, and J. P. Sterbenz, "Highly-dynamic cross-layered aeronautical network architecture," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, no. 4, pp. 2742–2765, 2011.

[9] T. H. Chung, M. R. Clement, M. A. Day, K. D. Jones, D. Davis, and M. Jones, "Live-fly, large-scale field experimentation for large numbers of fixed-wing UAVs," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1255–1262.

[10] A. Magnano, X. Fei, and A. Boukerche, "Predictive mobile IP handover for vehicular networks," in *2015 IEEE 40th Conference onLocal Computer Networks (LCN)*. IEEE, 2015, pp. 338–346.

[11] S. Sendra, A. Rego, J. Lloret, J. M. Jimenez, and O. Romero, "Including artificial intelligence in a routing protocol using software defined networks," in *Communications Workshops (ICC Workshops), 2017 IEEE International Conference on*. IEEE, 2017, pp. 670–674.

[12] S. Ali, J. Qadir, and A. Baig, "Routing protocols in delay tolerant networks - a survey," in *2010 6th International Conference on Emerging Technologies (ICET)*, Oct 2010, pp. 70–75.

[13] A. Vahdat and D. Becker, "Epidemic routing for partially connected ad hoc networks," Duke University, Tech. Rep., 2000.

[14] L. Gao, S. Yu, T. H. Luan, and W. Zhou, *Delay Tolerant Networks*. Springer, New York, NY, USA, 2015.

[15] H. Kang and D. Kim, "Vector routing for delay tolerant networks," in *2008 IEEE 68th Vehicular Technology Conference*, Sept 2008, pp. 1–5.

[16] J. P. Rohrer, "Geographic centroid routing for vehicular networks," in *Proceedings of the Seventh International Conference on Advances in Vehicular Systems, Technologies and Applications (VEHICULAR)*. Venice, Italy: IARIA, June 2018. Available: https://cdn.jprohrer.org/documents/publications/Rohrer-2018.pdf

[17] A. Lindgren, A. Doria, and O. Schelén, "Probabilistic routing in intermittently connected networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 7, no. 3, pp. 19–20, July 2003. Available: http://doi.acm.org/10.1145/961268.961272

[18] M. Musolesi and C. Mascolo, "Car: Context-aware adaptive routing for delay-tolerant mobile networks," *IEEE Transactions on Mobile Computing*, vol. 8, no. 2, pp. 246–260, Feb 2009.

[19] J. P. Rohrer and K. M. Killeen, "Geolocation assisted routing protocols for vehicular networks," in *Proceedings of the 5th IEEE International Conference on Connected Vehicles (ICCVE)*, Seattle, WA, September 2016, pp. 1–6. Available: https://cdn.jprohrer.org/documents/publications/Rohrer-Killeen-2016.pdf

[20] J. Lemley, S. Bazrafkan, and P. Corcoran, "Deep learning for consumer devices and services: Pushing the limits for machine learning, artificial intelligence, and computer vision." *IEEE Consumer Electronics Magazine*, vol. 6, no. 2, pp. 48–56, 2017.

[21] S. R. Gomes, S. G. Saroar, M. Mosfaiul, A. Telot, B. N. Khan, A. Chakrabarty, and M. Mostakim, "A comparative approach to email classification using naive bayes classifier and hidden markov model," in *2017 4th International Conference on Advances in Electrical Engineering (ICAEE)*. IEEE, 2017, pp. 482–487.

[22] A. P. Silva, K. Obraczka, S. Burleigh, and C. M. Hirata, "Smart congestion control for delay-and disruption tolerant networks," in *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 2016, pp. 1–9.

[23] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, 1989.

[24] D. Silver, A. Huang *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[26] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013. Available: https://doi.org/10.1177/0278364913495721

[27] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," in *Advances in neural information processing systems*. Morgan Kaufmann, 1994, pp. 671–678.

[28] R. Li, F. Li, X. Li, and Y. Wang, "Qgrid: Q-learning based routing protocol for vehicular ad hoc networks," in *2014 IEEE International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2014, pp. 1–8.

[29] A. Keränen, J. Ott, and T. Kärkkäinen, "The ONE Simulator for DTN Protocol Evaluation," in *SIMUTools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. New York, NY, USA: ICST, 2009.

[30] *NS-3 Manual*, NS-3 Consortium, 2016. [Online]. Available: https://www.nsnam.org/docs/release/3.26/manual/singlehtml/index.html

[31] M. J. F. Alenazi, Y. Cheng, D. Zhang, and J. P. G. Sterbenz, "Epidemic routing protocol implementation in ns-3," in *Proceedings of the 2015 Workshop on Ns-3* (WNS3 '15). New York, NY, USA: ACM, 2015, pp. 83–90. Available: http://doi.acm.org/10.1145/2756509.2756523

[32] *NS-3 Models Library*, NS-3 Consortium, 2016. [Online]. Available: https://www.nsnam.org/docs/release/3.26/models/singlehtml/index.html

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California