The Thesis committee for Justin Wayne Enderle

Certifies that this is the approved version of the following thesis:

A Routing Architecture for Delay Tolerant Networks

Approved by
Supervising Committee:

Christine Julien, Supervisor

Sriram Vishwanath

A Routing Architecture for Delay Tolerant Networks

by

Justin Wayne Enderle, B.S.

Thesis

Presented to the Faculty of the Graduate School

of The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

The University of Texas at Austin

May 2011

Acknowledgments

I would like to thank all of those who assisted me during the journey of creating this thesis. I would first like to thank my advisor, Dr. Christine Julien. I truly enjoyed the opportunity to know and work with her. Her always helpful advice, unwavering support, sincere understanding as well as her constant support and encouragement made this thesis possible. I would also like to thank Dr. Sriram Vishwanath, my co-advisor, as his perspective and support with the mathematical perspective of wireless networking was very helpful.

It was a pleasure for me to have known and worked with the fellow students of the Mobile and Pervasive Computing Lab. I cannot imagine a better environment to have been a student, with my fun and extremely bright lab mates Seth Holloway, Tony Petz, Drew Stovall, Vasanth Rajamani, Taesoo Jun, Nicholas Paine, and Nirmalya Roy providing ideas and help with all sorts of problems, technical or otherwise. Thank you for not only making graduate school educational, but also fun.

I cannot overstate my gratitude to Heidi Ruffner, my manager at Sandia National Laboratories. Without her unwavering support, encouragement, and insightful comments, successfully completing this after joining the workforce full-time would not have been possible. It is truly special to have someone in your corner cheering you on for success, and doing what they can to help you achieve it.

I would also like the thank the University of Texas at Austin for providing

me with the MCD and Thrust Fellowships. Without their financial support, my studies and UT-Austin would not have been possible.

I would lastly like to thank my family. Without my parents, Kevin and Jeanne, I would never have had the work ethic and confidence to push myself and strive for higher and higher goals. How much their support means to me, through success and failure, cannot be put into words. I would finally like to thank my wife, Rachael, for being there through this journey with me, through the good and the bad. Her love, support, and companionship made the day to day life a true joy, even through the stress and challenges of writing this thesis.

Justin Wayne Enderle

The University of Texas at Austin
May 2011

A Routing Architecture for Delay Tolerant Networks

Justin Wayne Enderle, M.S.E.
The University of Texas at Austin, 2011
Supervisor: Christine Julien

As the field of Delay Tolerant Networking continues to expand and receive more attention, a new class of routing algorithms have been proposed that are specifically tailored to perform in a network where no end to end paths between devices are assumed to exist. As the number of proposed routing algorithms has grown, it has become difficult to fully understand their similarities and differences. Although published results clearly show different performance results between algorithms, it can be difficult to pinpoint which of their characteristics are most responsible for their performance differences. This thesis proposes an architectural framework to define the underlying features that Delay Tolerant Network routing algorithms are composed of. Popular routing algorithms from research are discussed and shown to be compositions of the proposed architectural features, thereby validating the architecture itself. The architectural framework is also shown to be a useful guide to developing a modular and configurable simulation platform. Algorithms from literature were implemented as a composition of features, which can easily be modified and combined later to define and implement new algorithms. Better understanding the underlying structure and similarities between different routing algorithm approaches is key to truly analyzing their performance and obtaining a deep understanding of which components of an algorithm have the most influence, both positively and negatively, on the results. Armed with this knowledge, designers of Delay Tolerant Networks can more easily determine the proper composition of routing algorithm features to best fit their needs.

Contents

Chapter 1

Introduction

Mobile computing and networking have been very active areas of research for some time. Traditionally, the research in these areas has focused on treating the networks formed by mobile devices similarly to traditional, wired networks, with the addition of techniques used to help mitigate network failures caused by lost links. The research community approached these networks as mostly connected, and embedded that assumption into the approaches that were used to facilitate networking. Recently, the field of Delay Tolerant Networking (DTN) has been the subject of increased attention from the academic community. Delay Tolerant Networking research makes a fundamentally different assumption about its network, which is that it is inherently disconnected and unreliable. A Delay Tolerant Network (Figure 1.1) is unreliable to the point that direct communication paths are *never* assumed to exist between the source and destination of a data transmission. Unfortunately, removing the connectivity assumption nullifies the direct use of a large field of mobile networking research that relies on explicit path availability. A large amount of DTN research, including this work, is dedicated to developing and understanding networking techniques not reliant on explicit paths.

Building a network designed to anticipate and mitigate large delays and mul-

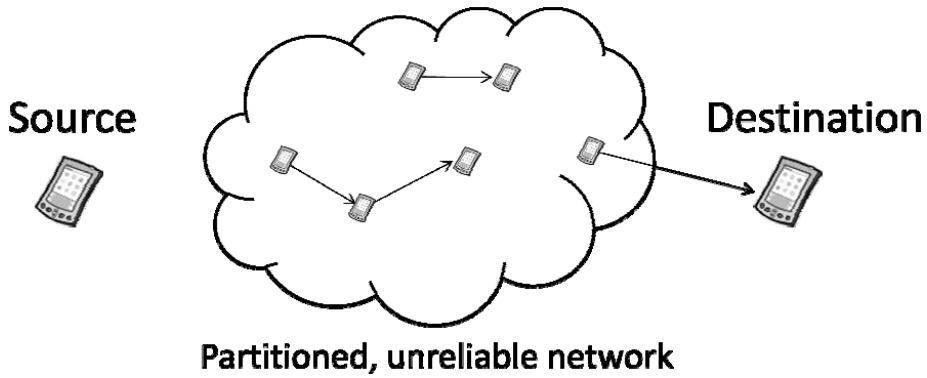


Figure 1.1: Delay Tolerant Network Model

tiple disconnects opens up new and exciting applications previously not possible. One application of Delay Tolerant Networking is to provide limited communication in extreme environments. In an extreme environment the communication devices form a very sparse network, with devices often completely isolated and out of range from one another. The capability to exchange useful information between isolated devices provides an enabling technology for DTNs. For example, providing opportunistic networking to remote villages [36] can provide them with crop prices from market centers traditionally beyond communication, allowing them the opportunity to adjust when crops are brought to market in order to obtain the best price possible [8]. Further examples of extreme environment DTNs include collecting sensor information from remote wildlife [27, 41], connecting nomadic tribes [15], as well as creating large scale underwater networks that communicate with low bandwidth acoustic transmitters [21, 46]. In these types of extreme applications, any path oriented networking approach will surely fail, therefore alternative and unconventional techniques must be employed to transmit the data across large regions with few resources. One common approach in these environments is to use message ferries [52]. A message ferry is a mobile object such as a person, bicycle, or bus that is already

travelling across large distances and encountering many other regions along its way. By simply giving those existing resources the ability to buffer and transmit data, information can be exchanged between other devices in potentially remote locations that they encounter along their route. The key with this approach is to take advantage of existing resources in the environment, requiring as little new infrastructure as possible.

Another application of Delay Tolerant Networks is to make use of intermittent and unreliable communication opportunities to expand the access and capabilities of existing networks. In contrast to an extreme environment, this approach attempts to increase the communication capabilities of traditional networks by using the opportunistic communication used by DTNs that is typically ignored by reliable networking systems. Although in many areas existing communication infrastructure such as wireless 3G or WiMAX already exists, their capacity and availability is limited. DTNs can increase access by offering limited communication capability to devices that were previously considered too unreliable to participate. Bandwidth advantages come from using a Delay Tolerant Network to transfer traffic that can tolerate delays and unreliable devices. This traffic will then bypass the infrastructure by using peer to peer communication, thus reducing the load on the infrastructure. Opportunistically exchanging information for localized social networks [23] and vehicular networking [17, 51] are a few examples of using DTNs to expand access and increase performance in connected environments.

Although ignoring path-oriented networking techniques provides a number of new and exciting applications, it also introduces a number of challenges that must be addressed. Addressing schemes, operation across heterogeneous networks, and application design are a few of the additional challenges to be considered [19], many of which are addressed by the DTN Bundle Protocol [39]. The Bundle Protocol is more primarily focused on creating an overlay network that operates on top of

both reliable and unreliable networks. This overlay network provides the abstraction necessary to seamlessly transmit data across the different available network types instead of simply being restricted to one type or another. The Bundle Protocol is more concerned with how the application data will be packaged and how to transition between the different network types. It does not, however, address how the data will be transmitted across the unreliable Delay Tolerant Networks, which is instead a question of routing, the primary focus of this thesis.

Traditional strategies for routing used in mobile ad-hoc networks such as AODV [37] and DSR [26] are not practical for a DTN, as they explicitly rely on the ability to discover and continuously maintain a complete route between hosts [24]. True routing algorithms do not really exist for Delay Tolerant Networks, as a consistent path between the source and destination rarely exists. In practice, DTNs employ a variety of forwarding strategies to opportunistically transfer copies of messages between hosts as they become connected. As the copies are spread throughout the network, it becomes increasingly likely that a message copy will be forwarded by intermediate devices and successfully delivered to its destination. The mobility within the network helps ensure that new connections occur, thus spreading message copies across a variety of hosts. This "store and forward" technique is the primary mechanism used to transmit data in a DTN.

DTN routing algorithms all strive to deliver as many messages as possible, quickly, and with little overhead by minimizing message copies. Unfortunately, as is often the case, it is necessary to sacrifice the performance of one goal to enhance that of another. A large number of DTN routing algorithms have been proposed that strive for different goals and set different performance priorities. It is the job of a system designer to carefully weigh the requirements of the target application in order to determine the proper balance necessary to achieve its objectives. The large number of algorithms can make it difficult to choose which approach is cor-

rect for a particular application. This difficulty is exacerbated by the fact that the evaluation of routing algorithms is often limited by time and resources to a limited number of scenarios. Without personally evaluating algorithm performance using simulators or real world testbeds, a system designer is forced to rely solely on published results. The unfortunate reality is that published results often present a new algorithm's performance under a limited set of environment and against only a subset of the mainstream algorithm. These restrictions are inherent to the medium due to necessary paper length restrictions and authors' desire to post competitive results, potentially at the expense of another algorithm's results that outperforms the proposed algorithm.

This thesis is focused on developing a deeper understanding of routing algorithms for Delay Tolerant Networks. This work does not propose new routing strategies but instead proposes an architecture that can be used to better describe, build, and understand routing algorithms. The proposed architecture (Chapter 3) breaks down the various approaches into their fundamental features and illustrates that all DTN routing algorithms are really just different compositions of the same small set of strategies. By breaking down DTN routing algorithms into a building-block based architecture, we can begin to have a more systematic way of comparing and contrasting competing algorithms, making it easier to isolate the differences among approaches and evaluate them based on those differences alone. This routing algorithm architecture also provides a common vocabulary that can be used to describe the algorithmic features and is an important step to enhancing the discussion of DTN routing as a community. An additional contribution of this work is towards the development of flexible and expandable simulation environments. A configurable simulation platform was developed based on the underlying routing architecture (Chapter 4). Routing algorithm implementations are broken down into basic building blocks, which are then composed together to create the desired al-

gorithm. This architecture and simulation platform makes it easy for a developer to try new approaches while only modifying a particular feature, while also making it easy to incorporate those changes into other routing algorithms as well. A core set of popular routing algorithms were implemented with the simulation platform and then evaluated across a large number of different scenarios (Chapter 5). Finally, conclusions that can be drawn from this work and future considerations are presented in Chapter 7.

Chapter 2

Background

The following sections provide additional information related to Delay Tolerant Networking and the evaluation of routing algorithms. Section 2.1 details the performance metrics that are most used to understand DTN routing algorithm performance, as well as introducing the most popular algorithm types that have been proposed. Section 2.2 explores the importance of mobility in a Delay Tolerant Network and how this has led to development and use of mobility models that attempt to capture the underlying patterns in real world movement. Finally, Section 2.3 discusses the different network simulation platforms that were considered for use in the implementation of a DTN simulation framework following the routing architecture proposed by this thesis.

2.1 Metrics and Approaches

Three primary performance metrics are traditionally used to evaluate Delay Tolerant Networking routing algorithms: delivery success rate, speed of delivery, and the amount of overhead generated. Message delivery in a DTN is not guaranteed, hence the importance of understanding the percentage of messages successfully delivered.

Although a DTN by definition is required to tolerate delay, minimizing the delay experienced is still important, making delay another key metric of routing algorithm performance. The final primary performance factor considered is the amount of overhead incurred. In a DTN, overhead is primarily associated with the generation and spread of message copies throughout the network. Each time a message is exchanged and copied into another device's buffer, overhead is incurred. As overhead increases, the potential exists for buffers to overflow causing message copies to be dropped. Every message copy dropped by a buffer is a lost copy that can no longer be exchanged between hosts, thus hindering the ability of the message to reach its final destination. As the primary scheme for delivery in a DTN is often to create copies and distribute them to other hosts, buffer management is a concern of the highest order. A variety of schemes have been proposed with the goal of reducing unnecessary message replication while still maintaining tolerable delivery probability and latencies. This is achieved through restricted flooding techniques [42, 44], by exploiting the underlying mobility patterns [13, 32, 51], and by using erasure and network coding [48, 50]. Chapter 3 provides in depth details about the different approaches and how they fit into this work's proposed routing architecture.

2.2 Mobility

Communication in a Delay Tolerant Network is often facilitated by the mobility of the devices, as it is through their movement that previously unreachable devices come into range and allow new communication opportunities with a variety of hosts. A number of the routing algorithms proposed are agnostic towards the device mobility patterns [24, 44, 50]. However, as the DTN community has recognized the impact of mobility on performance, a number of algorithms have been proposed that attempt to enhance delivery and network efficiency by using the predictable mobility patterns in the environment to make better routing decisions [32,34]. When

attempting to forward a message towards its eventual destination, knowledge about existing mobility patterns can make the delivery process more efficient by allowing a source to provide more message copies to hosts that are more likely to encounter the destination. For instance, in a network with completely random and independent mobility, a node encountering two relay nodes should distribute a copy to each relay as there is no statistical advantage in favoring one relay over the other for delivery. However, if the nodes have a predictable element to their mobility and the source is aware of this, a smarter decision can be made to distribute a message copy to the relay with the higher probability of encountering the destination.

With mobility dependent protocols being developed, a wide variety of mobility models have been proposed to try and capture more realistic movement patterns based on anticipated scenarios. Many of the models attempt to capture the social patterns of human mobility, while others explore transportation systems and animal herd patterns. More realistic models of these types of mobility systems better incorporate the non-randomness that exists and the bias that individuals have to follow particular patterns. Developing models that better exhibit the underlying non-random elements of mobility is important to evaluating delivery algorithms based on mobility detection and prediction. Lindgren et al. [32] propose a community modeled as a set of grid positions that nodes switch between, with a home grid position assigned to each device as well as a central gathering grid common to all devices, with the assumption that nodes move primarily between their home and the central gathering place. The author's posit that model this resembles real-life village and communities with neighborhoods and social centers, as well as mobile sensor networks tracking wildlife movement with watering holes as the central gathering points and individual herds as the communities. The Working Day Movement Model [16] is a more sophisticated model with many of the same ideas, which also adds a time scale to switch between sub-mobility models and locations for home,

work, and evening activities. A variety of transportation methods are available between the locations as well. The Community Based Mobility Model [35] represents the social dynamics between hosts as a graph with weighted edges indicating the strength of interaction between hosts and determines movement between grid positions based on the affinity to interact with the hosts currently located there. Vellambi et al. [47] evaluates both the Fixed Route Campus Bus Model and Area-Based Random Waypoint Model. The bus model simulates a network of buses traveling between fixed points on a college campus, with data automatically transferred between buses when they are stopped at the same location. The Area-Based Random Waypoint model assigns a specific sub-area to each host within which the nodes move according to a standard random waypoint mobility model with no restrictions made on how the nodes' areas overlap. The Obstacle Mobility Model [25] models the obstructions that exist in the environment, and determines the paths used to move around them. The mobility patterns of zebras have been studied as well [27]. An alternative approach to mobility modeling is to collect real world mobility statistics for a network and build models that fit the patterns observed from the collected data. Traces have been taken at a variety of locations and then used either as direct inputs for mobility to a simulator, or used to generate a synthetic mobility model matching the characteristics of the trace data [9, 38, 51].

2.3 Network Simulators

Network simulation is the primary tool used by researchers in the DTN community to evaluate proposed protocols. As the mobile network community has grown, a number of different network simulators have been developed to aid analysis. Simulators such as ns2 [7], OPNET [10], GloMoSim [6], and OMNeT++ [3, 45] have all been used extensively throughout the research community for a variety of network types. As the DTN community has grown, the ONE [28] and DTNSim2 [24] simula-

tors have been developed specifically for this environment. For the purposes of this work, OMNeT++ was chosen as the simulator for performance evaluations. The DTN specific simulators were not chosen as they are relatively young projects lacking a significant development community and important features such as detailed PHY/MAC implementations, limiting the ability to accurately model wireless contention and noise which can have a significant impact on performance. The highly modular aspect of OMNeT++ was an important capability that enabled the design of the configurable DTN routing framework described in Chapter 4. In addition to its modularity, the optional INET framework [1] available for OMNeT++ provides strong support for mobile wireless networking including implementations of important and popular mobile networking protocols. The OPNET simulator, while being full featured and used significantly in the corporate world, was not chosen as it is not as popular within the academic community, largely due to its costly licensing fees which can inhibit the availability of any work developed with it. Ns2, while garnering large acceptance and familiarity with academic researchers, suffers from a lack of modularity and architectural integrity. It is intended for researchers to begin migrating to ns3 [2] which addresses these architectural issues, but it is still under development. OMNeT++ is being actively developed by a large community of researchers and supported by annual conference workshops, with new features and capabilities being made available on a consistent basis. This along with its modularity, ease of use, and strong support for mobility made it a good fit for the goals of this research topic.

Chapter 3

Routing Architecture

3.1 Introduction

The purpose of all DTN routing algorithms is to deliver messages reliably, quickly, and with minimal overhead. Although a large number of different approaches seem to exist, my research has led me to discover that these algorithms address the different challenges of routing by altering the behavior of only a small set of specific features. The alteration of just one of these elements can drastically alter the characteristics of an algorithm, which determines its potential applications. Section 3.2 breaks down the fundamental architecture behind DTN routing algorithms, describing different elements that can be integrated together to determine the final characteristics of a routing algorithm. Section 3.3 then summarizes some of the more popular routing algorithms and illustrates how these algorithms are compositions of the features described in Section 3.2. This chapter is focused at an architectural level, with a description of a flexible implementation of this architecture discussed in Chapter 4.

3.2 Routing Features

A core number of features and approaches are used to define how a DTN routing algorithm performs. Choosing the correct features and their implementation is crucially important to obtaining a routing algorithm suitable for the target environment. Some features are applicable to specific needs, while others are universal and pertain to essentially any environment. The following features make up the DTN Routing Architecture components proposed by this thesis.

- **Buffer priority** plays an important role in determining the order in which messages are exchanged between hosts as well as which messages are dropped when buffer space becomes limited. When an encounter between two hosts is relatively short, it is not possible for the two hosts to exchange all of the messages in their buffers with each other as the time it takes to transmit all the information is shorter than the time they are in range. In this scenario, buffer priority is important as it determines which of the available messages will be transmitted. This has a significant impact on the deliverability of those messages as they are more likely to propagate to other hosts and throughout the network. Buffer priority also determines which messages are dropped when buffer space becomes limited. If more message copies are received than can be stored, the stored messages must be prioritized and low priority messages removed. Dropping a message from a buffer means it can no longer be exchanged with other hosts during future encounters, lowering its likelihood for successful delivery.

  Buffer priority can be set according to standard queuing techniques such as First In, First Out (FIFO). Additional queueing options use the length of time since a message was created, how many hops a message has travelled, or the number of copies of a message that have been distributed to determine priority.

A completely random selection scheme is also possible. All algorithms define this routing element, however its impact on performance will vary depending on how limited the buffer space is and how long the encounter times are. With sufficiently long encounter times and essentially unlimited storage space, the buffer priority scheme plays a much lesser role in determining algorithmic performance.

- **Mobility prediction** is a method that attempts to take advantage of predictable mobility patterns among the network devices in order to increase the efficiency of message distribution. When predictable mobility patterns exist, a smarter decision can be made to only distribute message copies to relays with a higher probability of encountering the destination. There are two defining characteristics to mobility prediction; how the nodes learn the mobility pattern and how that mobility knowledge is used to influence message distribution. Different approaches exist for the devices to learn and/or predict mobility patterns. For very systematic deployments where the devices follow a predetermined schedule (e.g., city buses, satellite orbits), the mobility pattern can be preprogrammed into the devices. For less systematic networks the mobility is often inferred by observing and storing the frequency of contact between hosts. Each host keeps track of how often it encounters the other hosts, providing a local estimate. This estimate is constantly updated as time goes on, allowing the mobility predication to adapt to changing mobility patterns in the environment. To gain a larger perspective on network patterns, hosts can exchange their local mobility observations with each other. For instance, knowing how often a neighbor encounters a particular destination is useful to determine how well it could be used to deliver messages to that destination. Once an estimate of mobility has been determined, the routing algorithm uses it to influence forwarding decisions. If the algorithm has high confidence in

the accuracy of its mobility prediction, then message copies can be forwarded exclusively to the relays that have been predicted to frequently encounter the destination. However, when the prediction is less accurate, additional message copies are forwarded to nodes with a lower likelihood of encountering the destination to provide for more opportunities for delivery in the event the mobility predictions are not accurate.

- **Message summary exchange** is a key feature of almost all algorithms and is used to increase the efficiency of message transfer. With message summary exchange, a list summarizing each host's buffer contents is exchanged during connection establishment. After receiving the other host's summary list, a host checks the received list against its buffer contents to ensure that it only attempts to exchange messages the other node does not already have. This additional coordination is a simple but important feature that drastically reduces the amount of unnecessary message copies distributed throughout the network. This feature does not restrict messages from being spread to and buffered in every node of the network, it simply makes the message exchange more efficient. In a DTN, connection opportunities are often rare and short, therefore any feature that makes better use of those opportunities is beneficial. The variability in this approach among algorithms is typically in how the summary information is represented and its accuracy. Longer summary exchanges can provide more information about buffer contents, but at the expense of additional overhead.

- **Message curing** is a routing element that gets its name from those who consider the opportunistic spreading of messages throughout a DTN to be similar to the spread of a disease. Once a message has been delivered to its destination, its message copies are no longer needed in the buffers throughout the network. After a successful delivery occurs, a "cure" message identifying the

delivered message is then spread epidemically throughout the network in the same manner as the original message. The cure notifies other nodes that the message has been delivered and that any copies of it can be removed from its buffers. Although this routing element incurs additional overhead in that the cure lists must be stored as well as exchanged upon each encounter, the buffer space reclaimed by this feature makes more space for copies of undelivered messages. Given that buffer space is often a primary constraint, this feature is useful in all but the most limited circumstances. How this feature is implemented is relatively straightforward, with the primary difference between algorithms being whether or not this feature is included.

- **Copy distribution restriction** is a common feature used to track and limit how messages are spread. A "copies" field is added to each message and, when first injected into the network, is initialized to a value that is used to limit the total number of copies of that message which may be created. The manner in which the copies field is updated and used to limit message distribution distinguishes the different copy restriction approaches from one another. One approach is to treat the initialized value of the copies field as the maximum number of copies of the message that may exist in the network. Each time a copy of a message is distributed to another host, the number of copies the source may distribute is decremented by the number of copies the other host is now allowed to distribute. For example, a message is initialized to a global maximum of 10 copies in the network. If the source sends that message to another host and allows it to distribute 4 copies, then the source's copies field is decremented to 6 and the copies field of the destination's message is initialized to 4. The total sum of the copies field across the entire network is still equal to 10. If the copies field is greater than 1 for any particular host, then that host is capable of generating new copies of the message for other

hosts. Once the field drops to 1, that host can no longer create additional copies of the message. At that point, the message can only be distributed to further hosts by removing the message from its own buffer and passing it to another device. This does not increase the number of message copies in the network, as one host deleted a message and another added one. Note that even if the copies field is greater than one for any particular message, it still only occupies one slot in the host's buffer. A message's copy limit is reached when the copies field has a value of 1 for every host across the network. Other approaches to copy restriction also initialize every new message with a copy limit, however this value is only tracked locally by each host. This provides a relative, not absolute limit on the number of copies for a particular message. For example, each node could be limited to only distributing 2 additional copies of the message. Copy distribution restriction is a fundamental routing element used to tweak the performance of a protocol. It is important to take advantage of the buffer resources that are available without overloading them.

- **Hop length restriction** is a routing element that limits the number of hops a message can travel to reach its destination. Similar to the copy limit restriction, a "hops" field is added to each message and initialized to the maximum number of hops a message may travel. The maximum hop length limits the number of intermediate relays that can be used to deliver a message. Each time a message is transferred to another host, its hop field is decreased by one. Once that value decreases to 1, the message can no longer be transferred to other nodes and is restricted to only being delivered directly to its destination. Setting a maximum hop length does not explicitly limit the number of copies of a message that are created throughout the network, it only limits how many hops from the original source the message can travel. For example, a hop limit of 2 will not be limiting if all hosts are within 2 hops from one

another. When the hop limit is carefully tuned to the specific application scenario and conditions, the restriction does have the effect of reducing the number of message copies. If the hop limit is too restrictive, then performance will suffer, especially for networks with highly localized mobility. Delivering a message across a large distance is difficult if it is limited to traversing a small number of mobile hosts which do not encounter many new nodes.

3.3 Routing Algorithms

In this section, a number of popular routing algorithms are described in terms of how they use the routing elements described in the previous section. The algorithms encompass a wide range of approaches, from the most basic Flooding to erasure and network coded schemes. This section shows that the proposed routing architecture provides a structured framework that can be used to break down the proposed algorithms into their core components.

Broader class labels are often used to group different algorithms together. These broader class labels are useful in that they provide an additional indication of how closely related the algorithms are. In all likelihood, routing algorithms belonging to the same class will be able to directly swap implementations of specific features among themselves. Algorithms from different classes, on the other hand, will have more fundamental differences in their underlying implementations that might restrict directly swapping implementations of specific features. Three broad class labels that can be used to illustrate this point are the non-coded algorithms, erasure-coded algorithms, and network-coded algorithms. Although all of the algorithms from these three classes can be broken down into how they compose the described routing elements, the approaches used within the same classes are much more likely to be very similar.

The following sections provide more detailed information on various DTN

routing algorithms. A more general discussion of each algorithm's properties is presented which is then followed with more details of specific implementations from the literature. The non-coded protocols of Flooding (Section 3.3.1), Epidemic (Section 3.3.2), PRoPHET (Section 3.3.3), Binary Spray and Wait (Section 3.3.4), and Binary Spray and Focus (Section 3.3.5) are discussed first, followed by the Erasure Coded (Section 3.3.6) and Network Coded (Section 3.3.7) algorithms. The distinguishing characteristics of each algorithm are also broken down and illustrated to be a combination of the architectural features that were proposed in Section 3.2, further proving that the routing architecture works well to understand and compare a wide variety of the algorithms.

3.3.1 Flooding

This is the most simplistic algorithm available. Every time a host encounters another node, they both attempt to send the entire contents of their buffers to each other. Basic bookkeeping is performed to ensure a host only sends a message to another host once. There is, however, no coordination between hosts to determine which messages should not be transferred because a node already has it or does not need it. Practically, this approach is rarely used, but it does provide a useful benchmark to determine the performance for the most unsophisticated case, and is often used for comparison in academic literature. If network bandwidth and buffer space are limited resources, which they often are, this method will utilize them least efficiently. If, however, both resources are unlimited, then this approach can result in low delivery latency, with a high delivery probability as every possible communication opportunity is fully exploited. Following the established architecture this algorithm only implements the buffer priority routing element. This serves as the basic foundation upon which all the other algorithms are built.

3.3.2 Epidemic

The Epidemic routing algorithm [44] improves upon Flooding by adding coordination between hosts to better determine which messages should be exchanged. Upon contact two hosts exchange summary vectors containing a list of messages they currently have in their buffer. This additional summary exchange prevents a node from transferring a message to a node that already has it, or knows that it does not need it. Epidemic routing performs the same unrestricted message distribution as Flooding but does so more efficiently. The performance characteristics will be slightly better because only useful messages are transferred. The original algorithm definition by Vahdat and Becker [44] also describes an optional limit to the number of hops a message can travel. This algorithm is a composition of FIFO buffer priority, message summary exchange, and an optional hop limit restriction component. Without the optional hop limit restriction, this algorithm is focused on increasing the efficiency of message exchanges, but not on limiting message replication when buffer space is a restricted resource.

3.3.3 PRoPHET

PRoPHET [32] stands for a "Probabilistic Routing Protocol using History of Encounters and Transitivity". This algorithm uses mobility prediction to limit the distribution of messages throughout the network. The base functionality builds from the Epidemic algorithm with only two modifications. First, mobility encounters are tracked and exchanged to calculate delivery probabilities between hosts. Second, a message copy is only distributed to another host it has a higher delivery probability than the current node does. For a network with deterministic mobility patterns, this algorithm will provide similar delivery probabilities and latencies while using fewer message copies to do so. If buffer space is limited, fewer message copies will result in more available buffer space, which will increase the overall throughput of the

network. Although this method reduces the number of copies created, an explicit copy limit restriction is not imposed. This algorithm is a composition of buffer priority, message summary exchange, and mobility prediction. An optional hop limit restriction is also considered, but the primary characteristic for this algorithm is the mobility prediction and its integration into the forwarding decision.

3.3.4 Binary Spray and Wait

Binary Spray and Wait [42] uses the copy limit restriction to balance acceptable latency and delivery rates. The algorithm consists of two different phases, with each phase operating according to a different set of rules. Each message begins in the "spray" phase, where it is initially assigned a copy limit restriction of L copies, with no restriction on the number of hops it can traverse. For each new node encountered during this phase, the encountered node will be given $\lfloor \frac{L}{2} \rfloor$ copies of the message while the source keeps $\lceil \frac{L}{2} \rceil$ message copies for itself. This is enforced by setting the "copies" field of the transferred message to the appropriate value. Once a host has only one remaining copy of a message that message enters the "wait" phase. During the wait phase, a message has no more copies to distribute to newly encountered nodes (i.e., $\lfloor \frac{L}{2} \rfloor = 0$) and it is restricted to only being transferred directly to the message's destination by a single hop (i.e. hop limit restriction). Analysis and simulation results [42] show that, under a random independent mobility assumption, this copy distribution approach reduces the amount of time necessary to distribute all of the message copies throughout the network. This algorithm is a composition of a buffer priority, message summary exchange, as well as copy and hop limit restrictions.

3.3.5 Binary Spray and Focus

Binary Spray and Focus [42] is similar to Binary Spray and Wait in that it consists of two phases, the first of which is identical the spray phase of Binary Spray and Wait. However, the second "focus" phase now uses mobility prediction for a more active approach. The hosts in the network track and exchange encounter times to determine a utility function which estimates the likelihood of encountering the other hosts. Whenever a host only has one message copy remaining, it enters the focus phase. Unlike the Binary Spray and Wait algorithm, a 1-hop limit restriction is not imposed during this phase. Instead, when another node is encountered during the focus phase, the message will be transferred to the encountered node and deleted from the source's buffer if the utility value for the encountered node is higher than its own (i.e., it has a greater likelihood of encountering the destination). Therefore, during the focus phase, the message is routed towards nodes with a greater likelihood of successful delivery while still maintaining the same number of copies since the sending node drops the message after forwarding it. If predictable mobility patterns exist in the network, this approach will result in lower delivery latencies, as messages in the second phase can actively continue to make progress towards their destination instead of waiting for one of the holding nodes to deliver it. This algorithm is a composition of the buffer priority, message summary exchange, mobility prediction, and copy limit routing elements.

3.3.6 Erasure Coding

Erasure coding [33, 49] is a technique that takes advantage of the diversity of forwarding paths available for a message. Erasure coding is used to split a message into blocks smaller than the original message, allowing more pieces of the message to be forwarded throughout the network as compared to a non-coded scheme. With erasure coding, the replication factor r is used by the source to determine how many

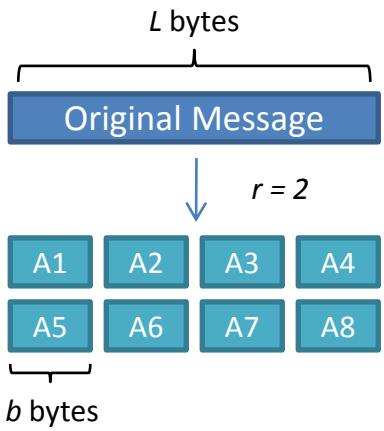


Figure 3.1: Erasure coding example with replication factor $r = 2$. $\frac{L}{b} * r$ blocks are created during encoding, with $\frac{L}{b}$ blocks required for decoding.

coded blocks are created. For a message of length L and a block size of length b, $\frac{L}{b} * r$ coded blocks are created at the source (Figure 3.1). All of the coded blocks generated by the source are then forwarded throughout the network towards the destination using the same concepts as non-coded algorithms. The statistical nature of erasure coding is such that once $\frac{1}{r}$ of the coded message blocks are delivered to the destination, the destination can decode the message. Receiving $\frac{L}{b}$ percent of the coded blocks of length b equates to receiving the same amount of data for a complete message copy, and with erasure coding these blocks can be used to recreate the original message.

A key concept for understanding erasure coding performance is that for the same amount of data replication, erasure coding can use a larger number of different forwarding opportunities to deliver its data. The performance of erasure coding is such that it eliminates both the worst case and best case scenarios [48]. The worst case scenario is avoided because spreading message fragments to a larger number of relays makes erasure coding less susceptible to the bad performance of individual relays in the network. In addition, erasure coding minimizes worst case delivery

latency by taking advantage of the multiple forwarding paths. Unfortunately, the best case performance also cannot be realized. The best case performance for a non-coded approach is dictated by how fast a *single* message copy can reach its destination. The best case performance for an erasure-coded scenario, on the other hand, is dictated by how fast a *complete set* of independently coded blocks can reach their destination. This invariably will take more time as it relies on more than one host to contact the destination.

Although with erasure coding the significance of an individual message has changed, the fundamentals behind message exchange remain the same in that once the coded blocks have been created by the source, the hosts buffer and forward them as if they were uncoded messages. Additional work is necessary for destinations to hold coded fragments in a reassembly buffer until there are enough available for decoding, but this is not a routing concern. The actual encoding/decoding implementation used does not matter. These coding issues remain independent of the other routing decisions, making it possible to directly re-use the same approaches used by non-coded algorithms.

The implementation of copy limit restrictions on erasure coding requires definition of two parameters. The first is determined during the coding phase, which uses the replication factor r to determine the number of unique coded blocks that will be created at the source. The second parameter determines the number of copies of each coded block that may exist in the network and determines the amount of overhead incurred by the forwarding approach.

After generating the coded blocks according to its replication factor r, the algorithm by Wang et al. [48] forwards an equal number of its coded blocks to each new host it encounters (Figure 3.2). Their algorithm also imposes a 2-hop limit restriction for all coded blocks which was carried over from another non-erasure coded 2-hop relay algorithm they analyzed [20]). The 2-hop restriction means each relay

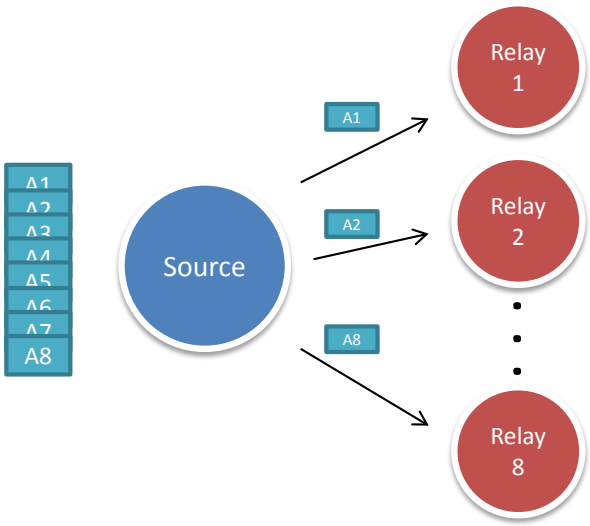


Figure 3.2: The approach by Wang et al. distributes an equal number of coded blocks to a predetermined number of relays.

can only forward blocks to their destination, and therefore prevents any additional copies of the coded blocks from being created, thus the overhead incurred during the initial erasure coding solely determines the amount of message replication that occurs. The algorithm proposed by Chen et al. [11] maintains the same 2-hop limit for each coded block, but allows an additional copy of each block to be created after the erasure coding phase. The first set of blocks are still distributed evenly to a specified number of relays as they are encountered. However, as many blocks as possible from the second set are sent to relays as they are encountered (Figure 3.3). The number of blocks aggressively forwarded from the second set is only limited by the contact duration between hosts. The second set of aggressively forwarded packets is intended to reduce latency, thereby increasing the best case delivery performance. Finally, for the approach described by Liao et al. [30], the coded blocks are not evenly distributed among a specified set of relays. Instead, the number of coded blocks that are passed to each relay is determined by their mobility prediction

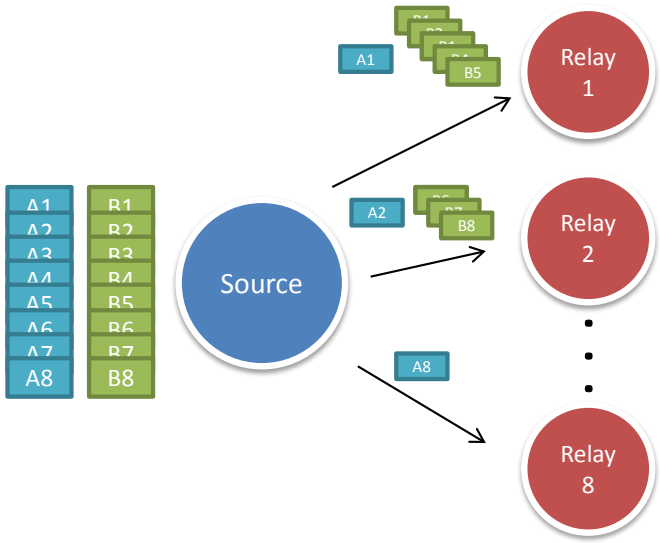


Figure 3.3: The second copy of coded blocks created by Chen et al.'s scheme is forwarded as quickly as possible to encountered relays. In this example, all of the second set of blocks (B1-B8) are distributed to the first and second relay.

mechanism, as the hosts with higher encounter probabilities will be given a larger percentage of the blocks to forward.

The fundamental characteristics that define the behavior of these erasure coded algorithms still adhere to the list established in Section 3.2. With the exception of the additional coding/decoding steps performed at the source and destinations, the algorithms are still distinguished by how they implement hop and copy limit restrictions, as well as mobility prediction. Interestingly, none of the previous approaches described address the issue of buffer priority in their approach. Even though it is not directly addressed, it is, as always, a defining characteristic of these algorithms.

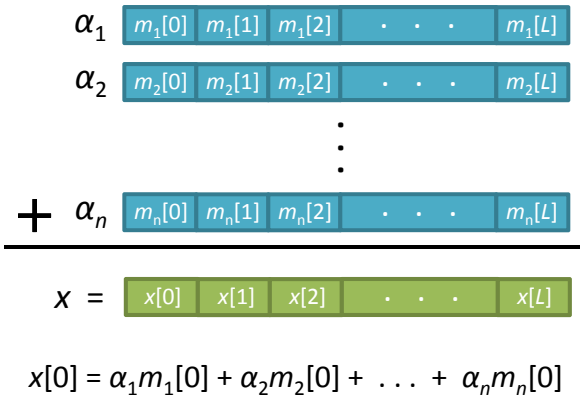


Figure 3.4: Creation of a network coded packet

3.3.7 Network Coding

Network coding [5] is a routing method that enables the intermediate nodes in the network to perform additional operations on the packets beyond the standard store and forward. With network coding, the packets exchanged between hosts are no longer entire copies of individual messages. Instead, the hosts in the network forward coded packets that are each a random linear combination of a set of the packets stored in the host's buffer. With this approach, each message exchanged contains some information about multiple packets instead of all of the information about a single packet. More formally, a network coded packet x is created by generating n random coefficients α_i from a finite field (e.g. $\mathbb{F}_{2^8}$) which are then linearly combined according to the formula $x = \sum_i^n \alpha_i m_i$ [12]. All of the addition and multiplication operations are performed within the chosen finite field.

The process of generating a single coded packet from n messages is illustrated in Figure 3.4. In this example, the finite field $\mathbb{F}_{2^8}$ is used, causing the encoding process to be performed on a byte-by-byte basis as this field only defines operations between within the range $[0, 2^8 - 1]$. Thinking of the messages as byte arrays of

length L, the first byte in the final coded packet x can be represented as $x[0] = \alpha_1 m_1[0] + \alpha_2 m_2[0] + \ldots + \alpha_n m_n[0]$. Alternatively, using a finite field of $\mathbb{F}_{2^1}$ would cause the encoding to be performed on a bit by bit basis. The reason a small finite field is attractive is that all the operations within that field can be pre-calculated and stored in a lookup table to increase efficiency. As the size of the finite field grows, the practicality of using a lookup table for all operations within that field becomes impractical, making each calculation much more costly.

$$\begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m,1} & \alpha_{m,2} & \cdots & \alpha_{m,n} \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_n \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{3.1}$$

As each coded packet that a host has can be represented as a single linear equation, all of the coded packets in a host's buffer can be thought of, and represented as, a system of linear equations. Multiple coded packets (x_i) and the linear equations that they represent ($\sum_i^n \alpha_i m_i$) are shown in matrix multiplication form in Equation 3.1. With this equation, the unknown variables are the uncoded messages m, with the coefficients α and encoded messages x being the known values. Storing the encoding coefficients in matrix form allows standard linear equation techniques to be used to encode and decode messages. Gaussian elimination is used to transform the coefficient matrix and the coded packets into row-reduced echelon form. Row-reduced echelon form makes it possible to easily determine the number of linearly independent rows of the matrix, which is also defined as its rank. Establishing the rank of the coded packets is important as it reveals any redundant information contained in the host's coded packets. If the row-reduced coefficient matrix establishes that a dependent equation exists, it implies that no additional useful information is obtained from that coded packet, and it can therefore be removed

without losing any useful information. In row-reduced echelon form, a decoded packet is represented by a single encoding coefficient of one, with zeros for all other row entries. This translates into the equation $m_i = x_i$, indicating that the coded packet x_i has been decoded into the actual message and can be delivered to its destination's application.

Network coding is distinguished from erasure coding not just by the type of encoding that is performed, but also by where in the network the encoding and decoding of messages takes place. With erasure coding, all of the coding and decoding is performed at the source and destination of a message, with intermediate nodes treating the coded blocks in a traditional store and forward manner. With network coding, the coding operations occur throughout the entire delivery path, with all of the nodes along the path involved in continuously coding, decoding, and re-encoding packets. When two hosts encounter one another, each packet they exchange is a newly coded packet generated by combining all the coded packets in their buffer with random coefficients. Although a seemingly infinite number of different coded packets can be generated by continually combining the buffered packets using different random coefficients, the number of *useful* coded packets that can be created from a single buffer of packets is limited by the rank of its coefficient matrix. Also, considering that each coded message a host receives will be inserted into its own coefficient matrix, coded packets are only useful to the other host if the equation that they represent is independent from the equations (coded packets) the other host already has. Therefore, the number of useful coded packets that can be exchanged between hosts is limited both by the rank of the sender and the rank of the receiver's coefficient matrix, as well as whether or not any independent packets can be generated between them.

The amount of memory and computation required to decode a coefficient matrix is determined by its size. The number of rows in the matrix is set by the

number of coded messages that a host has received, while the number of columns is set by the total number of uncoded messages that belong to the set. If all of the messages created in the network are allowed to be coded together, the amount of memory and overhead necessary can quickly grow as the number of columns increases with each new message in the network. To reduce the size of the coefficient matrices, messages can be grouped together into disjoint sets called generations [50]. Packets can only be encoded with other packets within the same generation, with each generation being assigned a separate coefficient matrix for encoding and decoding. Instead of a single large matrix, a host would manage many smaller matrices. Generation assignment can determine how the flows of traffic within the network can be combined. For example, one generation definition could assign all packets within the same source/destination pair together, with another approach basing generation assignment on the time period that a message was created. The method used to determine packet generation assignment is a new defining characteristic for network coded algorithms.

Routing strategies employing network coding are the most fundamentally different when compared to the other protocols. Network coded algorithms can still however, be described and differentiated by how they approach and implement architectural features previously discussed. However, unlike the previous algorithms, the implementations of addressing those features for network coding can usually not be directly applied to other non-network coded strategies.

With network coding, buffer priority and drop policy are only a concern if packets are subdivided into separate generations. Each coded packet that is forwarded is a random combination of all of the messages within its generation, therefore all messages within the same generation are treated with the same priority with some information from each contributing to the coded packet. The same concept applies when packets are dropped from a host's buffer. Instead of simply

dropping a single coded packet and its row from the coefficient matrix, the coded packet to be removed is first randomly combined with each of the remaining packets in the buffer. Because the removed packet is combined with all of the remaining ones before removal, choosing which packet to remove does not matter. If all of the messages are within a single generation, then buffer priority and drop policy is no longer a concern. However, when separate generations are used, a policy must be in place to give priority to one generation over another when packets are forwarded and when coded packets are dropped. Many of the same buffer priority and drop policy schemes can apply to generations just as they did for individual messages with other routing approaches.

Message summary exchange is also possible with network coding, but in a different way. Simply exchanging buffered message ID's is not possible, so instead some information to describe a host's coded messages must be exchanged. Exchanging the coefficient matrices between hosts can provide important details about what useful information can be exchanged. With another host's coefficient matrix, a host can check to see whether it can generate any coded packets that will increase the rank of the other host's coefficient matrix. If it can, then a useful exchange of information is possible and these useful coded packets are generated, otherwise no messages are exchanged. The message cure feature is also available for network coded algorithms, except that messages are cured at a generation level once all of its messages have been delivered, opposed to being on a message by message basis with other algorithms. The copy limit restriction is also present in that the hosts must determine how many coded messages it should exchange with other hosts. The approach by Widmer et al. [50] allows a host to generate a predetermined number of new coded packets to forward for every packet it receives that increases the rank of its coefficient matrix. The hop limit restriction is not applicable to individual messages, as it is not possible to track how far a message has travelled individually

once it has been encoded with other messages. However, it is still possible to limit how many hops coded messages from the same generation are allowed to travel. It is also possible, and has been proposed by Lin et al. [31], to use mobility prediction to help determine the number of coded messages exchanged between hosts. Finally, as was mentioned previously, the approach to generation assignment is another aspect in which network coded algorithms can be distinguished from one another.

3.4 Summary

The routing architecture that is proposed by this thesis has a lot of power to clarify the similarities and differences between different routing algorithms. Providing some structure to the analysis and debate of these algorithms should help the discourse over their benefits and tradeoffs be more productive. Each of the algorithms discussed in this section could be explained as a combination of the different architectural features, which further validates the applicability of the routing architecture. Additionally, the architecture can also provide a lot of benefit towards the implementation of DTN routing algorithms. The following section details the implementation of a DTN simulation framework that fully utilizes the underlying structure of DTN routing algorithms, which is shown to provide a lot of benefit in the implementation's structure and isolation of algorithmic components.

Chapter 4

Simulation Architecture

4.1 Introduction

One of the core challenges to understanding DTN routing algorithms is determining which protocol is appropriate in which situation. Routing algorithms tend to maximize their performance for specific operating conditions, and determining the algorithm that best suits the target environment is the ultimate goal of developing and deploying a DTN system. Unfortunately, it is often difficult to adequately determine which of the proposed algorithms will provide the best performance based on available literature. This difficulty exists because there simply are not enough published results and what exists is often not comparable and does not cover every scenario. Proposed routing algorithms often do not compare themselves against the best alternatives available, do not target your specific environment, or do not use realistic assumptions. Realistically, it is very difficult to determine the proper algorithm to use without evaluating it for your specific application.

The OMNeT++ simulation platform provides two key properties that are crucial to the development of a simulation architecture: modularity and configurability. The modularity comes from the core of OMNeT++'s software architecture

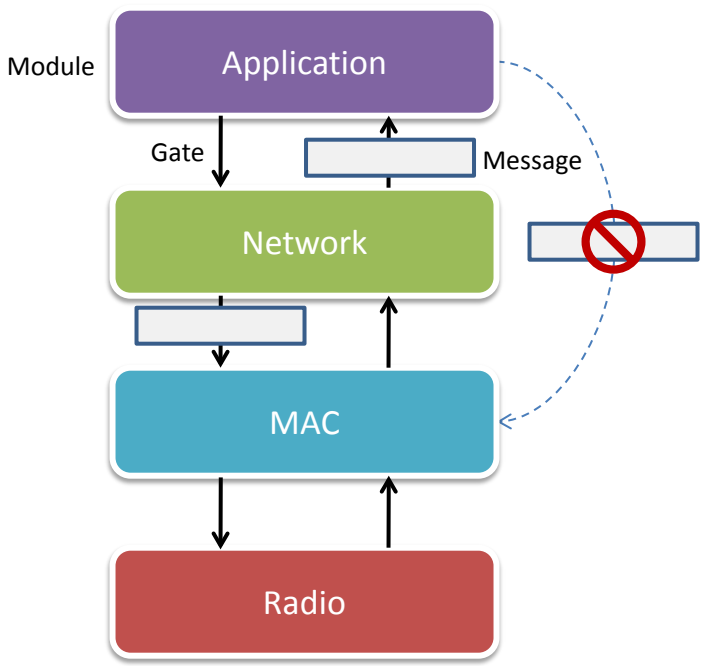


Figure 4.1: Generic network stack showing the relationship of modules, messages, and gates.

in that it isolates the various components of the system into separate software modules that can then be combined into more complicated systems. Each module is designed to perform a specific function, with a well defined interface through which it interacts with other system components. The modular architecture is achieved through three components, the *modules* themselves, the *messages* they exchange, and the *gates* through which messages are exchanged (Figure 4.1). A separate NED (NEtwork Description) language is used by OMNeT++ to define which modules are used, which gates are used and how they connect the modules, and any additional parameters that can be used to modify module properties. Figures 4.2 and 4.3 show how a few simple modifications of the NED definition can result in a substantially different network device. Naturally, a module itself can be composed of

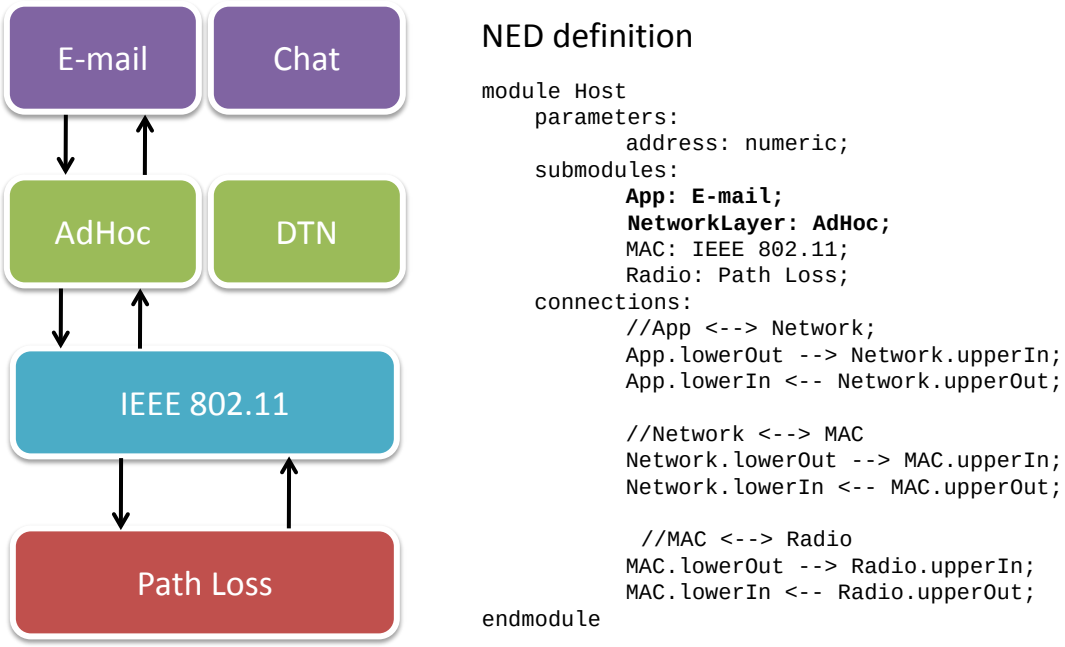
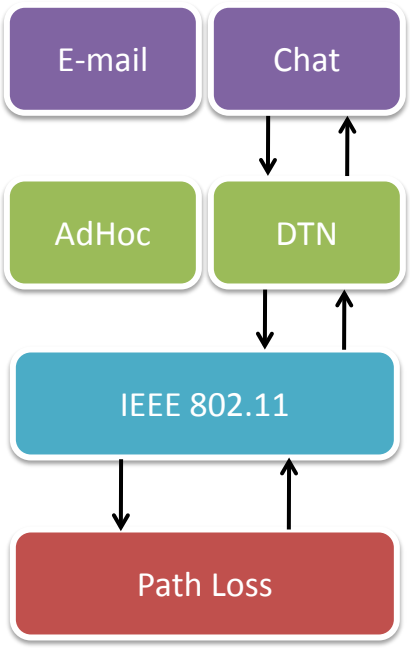


Figure 4.2: Network stack defining an AdHoc routing layer with an E-mail application.

multiple submodules which are then configured in the same manner. The smallest indivisible components that do not contain further submodules are implemented using C++. C++ allows the use of inheritance, encouraging further code reuse and the abstraction of common software components used by algorithms to minimize the code changes required to implement new DTN algorithms. The use of C++ also increases the likelihood to directly reuse much of the code for a real system implementation [29].

The configurability of OMNeT++ is achieved through two separate mechanisms. The previously described NED language is one way that allows previously defined software components to be combined in different ways. Just as important, however, is the use of separate configuration files to define the parameters for a simulation. Each OMNeT++ module can define any number of parameters that may



NED definition

```
module Host
    parameters:
            address: numeric;
    submodules:
            App: Chat;
            NetworkLayer: DTN;
            MAC: IEEE 802.11;
            Radio: Path Loss;
    connections:
            //App <--> Network;
            App.lowerOut --> Network.upperIn;
            App.lowerIn <-- Network.upperOut;

            //Network <--> MAC
            Network.lowerOut --> MAC.upperIn;
            Network.lowerIn <-- MAC.upperOut;

             //MAC <--> Radio
            MAC.lowerOut --> Radio.upperIn;
            MAC.lowerIn <-- Radio.upperOut;
endmodule
```

Figure 4.3: Modification of the App and NetworkLayer submodules configures a DTN host with a Chat application.

be individually modified for each simulation run without requiring any recompilation. Examples include buffer size, connection timeouts, and message hop limits. Even more interesting is using these parameters to define more complex aspects of the simulation such as the buffer priority scheme or even which routing algorithm is used. This external configuration greatly adds to the ability to manage and evaluate algorithms across a large number of scenarios.

As was extensively documented in Section 3, DTN routing algorithms are largely defined by how they implement a select number of features. The software architecture of the DTN simulator I have developed follows these principals, isolating the different features using OMNeT++ modules and inheritance. Furthermore, the configurability of OMNeT++ is leveraged, making it possible to reconfigure which

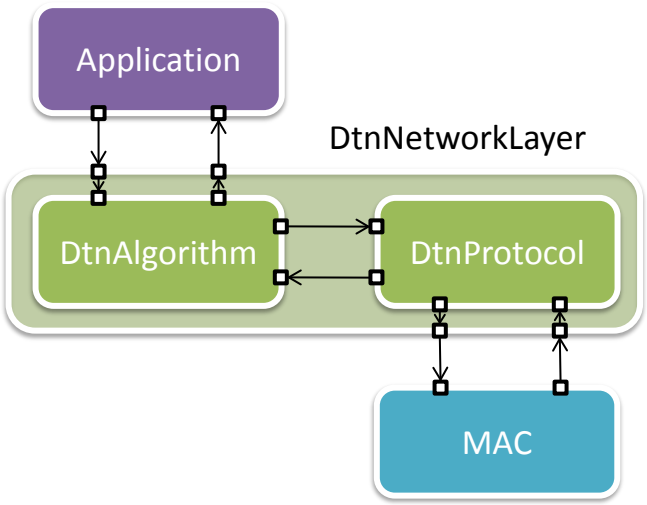


Figure 4.4: DtnNetworkLayer is composed of two submodules that are hidden from the external interface using internal gate connections.

routing protocols are used and their parameters through external configuration files. The following sections describe the overall structure of this DTN simulation framework, illustrating how its design fully leverages the discussed routing architecture to enhance its ability to develop and evaluate DTN routing algorithms.

4.2 DtnNetworkLayer

The Network Layer of the DTN simulator is composed of two simple submodules with C++ implementations, `DtnProtocol` and `DtnAlgorithm` (Figure 4.4). Externally, `DtnNetworkLayer's` interface consists of two upper and lower gates used to send and receive from the Application and MAC layers respectively. The NED language allows the `DtnNetworkLayer` to be defined such that these external gate interfaces are forwarded to the hidden internal submodules, making for a more flexible architecture composed of multiple submodules without burdening any outside

components with these implementation details. As the submodule names suggest, the functionality of the network layer implementation has been split between the algorithmic and protocol components. In a communication system, the protocol defines the specific rules that must be followed to participate in the system. This outlines the specifics for how hosts must coordinate to enable connection establishment, message transfer, and connection tear-down. An algorithm, in contrast, is more of a general approach to solving a problem. In general, academic research tends to focus more toward algorithms, and not protocols. Papers do not tend to focus on understanding and evaluating how hosts establish a connection with one another, whether there are acknowledgments for each message, or how lost connections are detected and handled. Algorithms deal with which messages are transferred between hosts and when and are concerned with the less straightforward challenges of maximizing the usage of the limited resources in the system such as buffer space and connection opportunities. Both the protocol and algorithmic components are necessary to accurately simulate a DTN system, but they are mostly independent from one another. Our routing implementations have followed this observation, and are described in detail in the following sections.

4.2.1 DtnProtocol

This module is focused on the mechanics of delivering messages between hosts, but not with making any decisions on which messages should be transferred and when. One of its primary responsibilities is connection establishment. This module periodically broadcasts beacons advertising its network and MAC address, which when received by other hosts are used to either initiate or maintain an existing connection. The current implementation is connection-oriented, in that a handshaking process must occur before application messages are transferred. Once a host receives a beacon, it responds with a unicast connection request to the source of the beacon. Once

the source of the beacon has received a reply, a connection has been established between the hosts and they may begin transferring messages. This handshake process is meant to ensure that bidirectional communication is possible, which is necessary to ensure proper acknowledgment of message delivery. Studies have shown that it is not safe to assume that communication from $A \rightarrow B$ implies communication is possible from $B \rightarrow A$ [14]. All communication between hosts is also currently acknowledged, which is important when a message is transferred from one host to another and not simply copied, as in the focus phase of the Binary Spray and Focus algorithm (Section 3.3.5). Without acknowledgment, it would be possible for the source to assume the destination received a packet when in fact it did not, resulting in that packet being dropped from the source's buffer without the corresponding insertion into the destination's buffer, potentially resulting in the complete loss of a message. Once a connection has been established, `DtnProtocol` also checks for periodic reception of beacon messages from the host to maintain the connection. If a beacon has not been received after a specified amount of time the connection is closed, and the handshake process must be followed again to reestablish communication. Certainly different approaches can be taken in regards to connection establishment and tear-down to strike the right balance between reliability without wasting precious connection time. The correct approach is likely dependent on the specific environment that is being operated in, and any evaluation to determine the right approach to these concerns is isolated within the `DtnProtocol` module and independent from the particular routing algorithm that is used.

The interface between `DtnProtocol` and `DtnAlgorithm` is a simple one. Once `DtnProtocol` has established a connection with another host, it calls `DtnAlgorithm`'s `startTx()` interface function. Once that has occurred, `DtnAlgorithm` will either create a summary message or select a message from its buffer to send to the other host, which is sent to `DtnProtocol` through the two modules' port connection.

DtnProtocol then receives the message from DtnAlgorithm through its port connection, where it then handles the transmission and potentially retransmission of that message until an acknowledgement has been received. After DtnProtocol has established that the message was successfully delivered, it notifies DtnAlgorithm through its ackMessage() interface function so the algorithm can take the appropriate action, which usually consists of retrieving another message from its buffer and sending it to DtnProtocol for delivery. After DtnProtocol determines a connection has been lost, it notifies DtnAlgorithm through its endTx() interface function. Multiple connections may be active at any one time, and the messages forwarded to DtnProtocol by DtnAlgorithm are an interleaved mix of the messages being sent to the different active connections. That is the extent of the limited interface between the two modules. Any alternative implementation of the DtnProtocol or DtnAlgorithm modules may be substituted so long as they follow these interface guidelines. This well defined interface and separation of concerns certainly adds to the modularity of the architecture.

4.2.2 DtnAlgorithm

This module's focus is on the implementation of the actual DTN routing algorithms, such as those described in Section 3.3. Figure 4.5 shows the basic class inheritance structure which is used for the implementation of the Flooding, Epidemic, PRoPHET, Binary Spray and Wait, Binary Spray and Focus, Erasure Coding, and Network Coding algorithms. DtnAlgorithm makes heavy use of C++'s inheritance as well as advanced data structures and encapsulation to maximize the reuse of basic routing features across multiple algorithms.

The DtnAlgorithm base class establishes the basic functionality for buffer management and the mechanics to retrieve messages for active connections based on some definition of priority. The Standard Template Library (STL) [4] map data

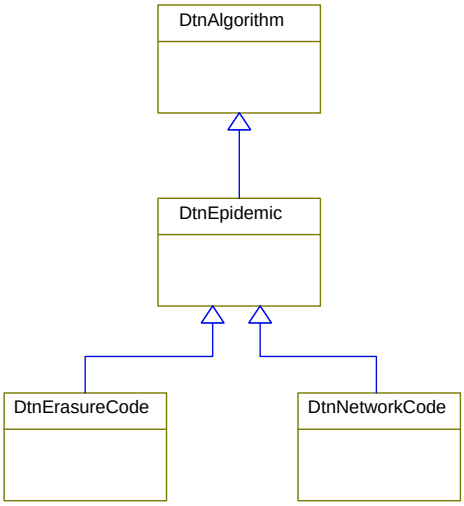


Figure 4.5: UML inheritance diagram illustrating the classes used to implement the `DtnAlgorithm` OMNeT++ module

structure is used to efficiently store and prioritize the messages that are currently being stored in a host's buffer. The STL implementations allow the specification of a custom comparator, which is used to determine the order in which elements are stored in the map. The ordering of messages in the map is what establishes buffer priority. Two examples of custom comparators are provided showing the FIFO (Algorithm 1: `msgCompareFifo`) and lowest hop count (Algorithm 2: `msgCompareHops`) buffer priority schemes. When a connection is established, the map is traversed in the order defined by the comparator to find messages to send to the other host. When a message must be dropped from the buffer, the ordering of the map makes it simple in that the last message in the map has lowest priority and is therefore chosen to be dropped. The comparator mechanism is a simple, flexible, and isolated method of defining buffer priority. In addition to maintaining the map that keeps track of all of the messages in a host's buffer, the `DtnAlgorithm` class also manages information for all of the hosts it has encountered so far. This is necessary to keep track of which messages have already been delivered to each host, and to determine

Algorithm 1: msgCompareFifo(DtnMsg msgA, DtnMsg msgB)

```
/* Message that arrived first has higher priority */
/* Returns true if msgA has higher priority than msgB */
if msgA.arrivalTime() < msgB.arrivalTime() then
    return true;
else if msgA.arrivalTime() > msgB.arrivalTime() then
    return false;
else
    /* Break ties using message ID */
    if msgA.ID() < msgB.ID() then return true;
    else if msgA.ID() > msgB.ID() then return false;
    else return false;
end
```

what message should be sent next. This is also where another host's message summary is stored to allow easy referencing of which messages each host does not need. Additionally, `DtnAlgorithm` implements the interface with the host's application to insert newly created messages into its buffer and allowing the network layer to deliver received messages to the application when appropriate.

The `DtnEpidemic` class provides an implementation of the message summary exchange, message curing, mobility prediction, copy limit, and hop limit restrictions that were described in Section 3.2. The Flooding, Epidemic, PRoPHET, Binary Spray and Wait, and Binary Spray and Focus algorithms are all implemented within this class as their differences are defined by simple and isolated modifications to how they implement these features. The `DtnEpidemic` class is parameterized to configure what types of information are exchanged after a connection is established. Message summary exchange, message curing, and mobility prediction all involve some extra information being exchanged at the beginning of an encounter, and each feature can be turned on and off individually. The same mechanism is used to exchange any additional information required by all of these features, therefore if any of these three features is enabled, a single summary message containing all of the necessary

Algorithm 2: msgCompareHops(DtnMsg msgA, DtnMsg msgB)

```
/* Message that has travelled fewest hops has higher priority */
/* Returns true if msgA has higher priority than msgB */
if msgA.hops() < msgB.hops() then
    return true;
else if msgA.hops() > msgB.hops() then
    return false;
else
    /* Break ties using message ID */
    if msgA.ID() < msgB.ID() then return true;
    else if msgA.ID() > msgB.ID() then return false;
    else return false;
end
```

summary information for the enabled features is exchanged after the `startTx()` function has been called by `DtnProtocol`. This makes it easy to add additional types of information that can be exchanged when two hosts meet. Further details on the architecture and of these features and their implementation in `DtnEpidemic` are outlined in the following list.

- **Message Summary Exchange** - To facilitate this feature, each host maintains a STL hash table containing the IDs of all of the messages that should not be sent to it during an encounter with another host. This list contains the IDs of all of the messages currently in its buffer or those that have already been delivered to its application. If message summary exchange is enabled, the two hosts will exchange their summary tables during the initial phase of the connection and store them for the duration of the connection. While exchanging data messages, each host will check the received summary list before sending any message to ensure the particular message is actually necessary, otherwise it moves on to the next message according to its priority scheme.

- **Mobility Prediction** - A single mobility tracking and prediction mechanism has been implemented that is used by both the PRoPHET and Binary Spray

and Focus algorithms to influence their forwarding decisions. The current mobility prediction scheme is taken directly from the PRoPHET paper [32] and uses the exchange of basic encounter information to infer the likelihood of a host delivering a message based on how often it encounters the destination, as well as how often it encounters other hosts that encounter the destination. The mobility prediction information is made accessible to the algorithms through a simple interface call `getDeliveryProb(sourceHost, destHost)`, which returns a normalized probability of `sourceHost` encountering `destHost`. Additionally, `higherDeliveryProb(hostA, hostB, destHost)` is provided, which determines whether `hostA` or `hostB` is more likely to encounter `destHost`. Since mobility prediction typically is used to determine the proportion of message copies to exchange with another host, this interface is sufficient to satisfy that requirement. The details of calculating delivery probabilities are hidden through this interface, making it easy to substitute different probability calculation schemes without impacting the algorithms using mobility information. This isolation is important, as different approaches have been proposed for this mechanism [32, 34, 48] and allowing an easy means to replace this scheme really adds to the potential for evaluating DTN routing algorithms.

- **Message Curing** - This feature is optional for all of the protocols that have been described and is enabled through a parameter in a configuration file. Much like the message summary exchange, the `DtnEpidemic` class maintains a hash table containing the unique IDs of messages it knows to have been successfully delivered to their applications. A host will add entries into its own table when it delivers a message to its own application, or when it delivers a message to its destination host and receives an acknowledgement for it. When this feature is enabled, hosts will exchange their list of cure IDs along with any other summary information after connection establishment

by `DtnProtocol`. Afterward, they will each update their local list with any new entries they received. If they receive a new cure entry matching a message currently in their buffer, that message will be dropped from the buffer. Again, the implementation of this feature is independent of the others, and can easily be modified with little to no impact on additional elements of the model. The message cure implementation in `DtnEpidemic` is also directly reused by the `DtnErasureCode` class, however `DtnNetworkCode` requires a different implementation as messages are cured at the generational level, and not as individual messages.

- **Copy Limit Restriction** - As with the other features, copy limit restriction is implemented in a single location and isolated from the other algorithmic features. Before any message is sent to another host, its `copies` parameter must first be set. The calculation and initialization of this value is the sole determining factor in what type of copy limit restriction is imposed. For instance, if the copy limit policy is to locally restrict each relay to only creating a set number of message copies to other hosts, then the sent message's `copies` parameter is initialized to that value before it is sent, and the host's local `copies` parameter is decremented by 1 each time that message is sent. Once the host's local `copies` count reaches 1, it is only allowed to send that message to its destination or to another relay host if it removes the message from its own buffer after sending it. The latter situation occurs during the focus phase of the Binary Spray and Focus algorithm. On the other hand, if a global copy limit is used, the `copies` parameter of a message is initialized when it is first inserted into the host's buffer after receiving it from its application, with this initialized value becoming the global number of allowable copies for that message. To maintain the global limit, the host's local `copies` parameter for a message must be decremented by the same value the sent message's `copies`

parameter is set to. For example, with Binary Spray and Wait, a host will distribute half of its available copies to the encountered host by setting the sent message's `copies` parameter to $\lfloor \frac{\texttt{copies}}{2} \rfloor$.

- **Hop Limit Restriction** - This feature's implementation is quite simple. Each message's `hops` parameter is first initialized to 0 before it is inserted into the host's buffer. The `hops` parameter is then incremented by 1 each time that message is sent from one host to another. Before any message is sent, the `hops` parameter is checked to see if it equals the established hop limit, and if so the message is not forwarded to the encountered host. The hop limit restriction is entirely established by the external parameter in a configuration file, requiring no modification of the algorithms to modify its characteristics.

To further illustrate the differences between algorithms and the ease with which their distinguishing characteristics can be implemented, two primary components of the routing algorithm implementation are shown with Algorithm 3: `doSendMessage` and Algorithm 4: `modifySentMessage`. Two important decisions must be made during an encounter with another host: which messages can be sent (`doSendMessage`) and what modifications should be made to those messages (`modifySentMessage`). As `doSendMessage` shows, the differences in this step between Flooding, Epidemic, Binary Spray and Wait, Binary Spray and Focus, as well as PRoPHET entail a very small amount of code. However, this small snippet captures some of the most fundamental and illustrative differences between these algorithms. The modifications made to sent messages, as shown with `modifySentMessage`, impose restrictions on what the other host is able to do with the message. The `modifySentMessage` procedure shows how the number of copies the other host is allowed to distribute is set, as well as the message's hop limit. Again, this relatively small bit of code represents some very fundamental differences between these algorithms and is an excellent example of how different algorithms can be accurately

Algorithm 3: doSendMessage(dest, msg)

```
if checkMsgSent(dest, msg) then
    /* Don't send if msg has already been sent to dest */
    send ⇐ false;
else if msg.dest() = dest then
    /* Always send if dest is destination of msg */
    send ⇐ true;
else if msg.hops() = hopLimit then
    /* Don't send to relay if hopLimit is reached */
    send ⇐ false;
else
    /* Protocol specific checks */
    switch routingProtocol do
        case Flooding
        case Epidemic
        case BinarySprayWait
            /* Don't send if this is the last copy */
            if msg.copies() = 1 then
                send ⇐ false;
            else
                send ⇐ true;
            end
        case BinarySprayFocus
            /* Forward last copy if dest has higher delivery prob */
            if msg.copies() = 1 then
                send ⇐ higherDeliveryProb(dest, msg);
            else
                send ⇐ true;
            end
        case Prophet
            if msg.copies() = 1 then
                send ⇐ false;
            else
                /* Forward if dest has higher delivery prob */
                send ⇐ higherDeliveryProb(dest, msg);
            end
    endsw
end
```

Algorithm 4: modifySentMessage(localMsg)

```
/* Protocol specific modifications */
switch routingProtocol do
    case Flooding
    case Epidemic
    case Prophet
        /* Local copy limit restriction */
        spreadCopies ⇐ copyLimit
    case BinarySprayWait
        /* Global copy limit restriction */
        spreadCopies ⇐ ⌊localMsg.copies() / 2⌋;
    case BinarySprayFocus
        if localMsg.copies > 1 then
            /* In spray phase, distribute half of copies */
            spreadCopies ⇐ ⌊localMsg.copies() / 2⌋;
        else
            /* In focus phase, send only remaining copy */
            spreadCopies ⇐ 1;
        end
endsw
outMsg = localMsg;
outMsg.setCopies(spreadCopies);
outMsg.setHops(outMsg.hops +1);
localMsg.setCopies(localMsg.copies() - spreadCopies);
```

described by the proposed DTN routing architecture.

4.2.3 DtnErasureCode

The `DtnErasureCode` implementation takes full advantage of its similarity to the algorithms defined by the `DtnEpidemic` class, requiring relatively little additional code for its implementation. As mentioned in Section 3.3.6, the bulk of an erasure coding implementation follows from the Epidemic algorithm, with the coding and decoding that occurs at the source and destination being the most fundamental difference. The current `DtnErasureCode` implementation does not actually perform

the real encoding and decoding process as the goal was not to evaluate computational complexity or the performance of various coding algorithms, but rather their implications on delivery rate, latency, and overhead. Instead, each message is assigned an additional `numBlocks` parameter. When a new message is created by an application, parameters specifying the replication factor r and block size b are used to initialize `numBlocks` to the appropriate value of $\frac{L}{b} * r$, with L being the length of the message. Each host merely keeps track of the number of blocks that it has for a particular message and determines that it can be decoded when `numBlocks` $\geq \frac{L}{b}$. Some additional code is required to manipulate and check `numBlocks` as necessary, but the vast majority of the implementation is inherited from `DtnEpidemic`.

4.2.4 DtnNetworkCode

The `DtnNetworkCode` class is the most fundamentally different from the other protocols, requiring a more extensive reimplementation of the architectural features from the `DtnEpidemic` base class. The sending and receiving of all messages must now be coded and decoded, as opposed to just the source and destination as with Erasure Coding. Additionally, the concept of a message ID no longer exists for the coded messages in the buffers. Therefore, new mechanisms were required to store and maintain the coded messages and their encoding coefficients in matrix form to simplify the encoding and decoding process. Generations are also used, which is a network coding specific feature used to segment coded packets into separate groups, allowing the decoding matrix and thus the packet overhead for each generation to be smaller. The cure feature, along with copy and hop limit restrictions are then applicable and implemented at the generation level. The current implementation uses a `fieldExponent` parameter to determine the finite field ($\mathbb{F}_{2^{\texttt{fieldExponent}}}$) used for the coding scheme. Similarly to `DtnErasureCode`, the actual data contents of each message are not coded and encoded, only the headers are. The contents of the

messages themselves make no difference to the routing algorithms, only the network layer header information does. Not performing the encoding and decoding on the full data contents greatly reduces the computational load and overall duration of the simulations. Even though the implementation of this algorithm is substantially different from the others, the implementation itself is still broken up into the separate architectural features like all the other algorithms.

4.3 Additional Modules

As Figure 4.3 illustrates, there are more components to the simulation than the networking layer, which has thus far received the majority of the discussion. The following section briefly describes other modules that are necessary to complete the simulation architecture and enable the evaluation of DTN Routing algorithms within a fully functioning network stack.

4.3.1 Mobility Module

In addition to a networking stack, each host also has its own mobility module that is used to determine when and where the host moves throughout the period of the simulation. A timer is used to periodically update each host's position, which is then used to calculate the distance between hosts that is used for Signal to Noise Ratio (SNR) calculations within the radio model. If the mobility pattern for a host is independent of its surroundings and the other hosts, then it is entirely local and requires no external information or coordination. If there is some level of coordination in the movement algorithm, then an additional global mobility module is added to the simulation environment to provide the necessary coordination between hosts. Depending on the mobility model, the global module will have knowledge about its environment and the locations of the current hosts, using that information to drive their movements. Specific descriptions of both types of mobility models are

outlined in Section 5. The mobility model being used can be easily specified using a configuration file, requiring no recompilation to switch between them. Mobility models can also be assigned on a host by host basis, allowing multiple types of host mobility patterns to exist within the same simulation.

4.3.2 DtnApp Module

Currently, the simulation architecture is targeted towards research at the networking layer, with the `DtnApp` module abstracting all of the above layers into a traffic generator that simply injects packets into the network layer. `DtnApp` is responsible for creating new packets according to a configurable statistical interval, determining their destination among the available nodes in the network, and assigning each packet a unique identifier that is used by the network layer to distinguish packets from one another. Currently, the `DtnApp` traffic pattern randomly distributes packets to all other hosts in the network according to a uniform distribution at a configurable rate. This is a somewhat arbitrary traffic generation choice and is intended to provide a dynamic and challenging set of traffic characteristics for a routing algorithm. Due to the modularity of the simulation structure, it would be simple to substitute in a more complex or application specific traffic generator should one wish to do so. In addition, it is certainly possible to expand the upper layers to better represent more complicated protocols, such as Bundle [40].

4.3.3 ChannelControl Module

The `ChannelControl` global module, provided by the INET framework [1], is used to coordinate wireless communication throughout the simulation. It keeps track of all of the wireless traffic and provides the calculated noise level to each host allowing them to determine whether a current message satisfies the minimum SNR level required for it to be received and delivered to the MAC layer.

4.3.4 DtnStats Module

This global module is responsible for aggregating performance characteristics related to the DTN routing algorithms. Many statistics are tracked individually by each host, however this module provides the ability to track global properties by combining statistics from individual hosts. A list of aggregated statistics provided includes:

- Number of generated messages
- Number of delivered messages
- Average delivery latency
- Cumulative Distribution Function (CDF) of the latency of delivered messages
- Histogram of the number of copies made of each message
- Histogram of the number of hops travelled by each message

4.4 Summary

As this chapter illustrated, the routing architecture put forth in Section 3.2 can be directly translated into a software architecture for a DTN simulator. By taking advantage of that architecture, as well as additional similarities between algorithms, the majority of the simulation implementation can remain constant even between different routing algorithms. The DTN simulations are truly composed of many independent components, making out simulation platform very powerful and flexible. The next chapter introduces some results obtained from the simulation platform described in this section and provides validation for this modular approach to algorithm development.

Chapter 5

Evaluation

5.1 Introduction

This chapter introduces some results that were obtained using the simulation platform that was developed according to the proposed routing architecture. The evaluations presented are not meant to be exhaustive and should not be considered an in-depth treatment of the areas discussed. Instead, this chapter simply establishes that the simulation architecture was implemented and introduces some performance differences between the routing algorithms, with a particular focus on the impact of the proposed architectural features. The simulation setup is discussed, providing an explanation of the simulation parameters used and methods implemented to track the statistical validity of the results, which is then followed by an introduction to the mobility models that were used. Afterward, results associated with the copy limit, mobility prediction, cure exchange, and hop limit features are presented and discussed.

5.2 Simulation Setup

A total of 50 hosts were used for the simulations presented in this chapter. Each simulation run lasted a total of 25000 seconds. During the first 5000 seconds, no messages were generated or exchanged, allowing the hosts to exchange mobility prediction data and begin to determine the delivery probabilities of the various other hosts. After 5000 seconds, each host's application generated six new messages with random destinations, sending them to their respective host's routing layer for delivery. This created a total of 300 unique messages to be delivered throughout the simulation. Each host was given a buffer size of 50 messages, creating an environment with limited buffer space to provide a challenging scenario for the routing algorithms. The FIFO buffer priority was used for all but the Network Coding algorithm, which instead assigned highest priority to the generations in its buffer with the highest rank. For Network Coding, each generation consisted of six messages, with all of the packets generated by the same host belonging to the same generation. A 5000km x 5000km simulation area was used, with the hosts' maximum wireless radio range set to 150 meters. A simple path loss noise model was used to calculate the Signal to Noise Ratios for the wireless receivers, which is a purely distance based calculation with no random variation over time. The Levy Walk and Village mobility models were used, which are further discussed in Sections 5.3.2 and 5.3.1 respectively.

In order to calculate and control the statistical validity of the simulation results, an additional framework was developed to automate the execution and collection of simulation statistics. The OMNeT++ simulation framework was integrated with the Akaroa2 framework [18], which is a statistical package designed to collect and control stochastic simulation results. The framework provides the capability to set specific requirements on the precision and confidence of the results being obtained. Independent runs of each simulation scenario are made as many times

as necessary to achieve those requirements. Although previous work was done to integrate Akaroa2 with OMNeT++ [43], it did not have the capability to manage statistics gathered from independent runs, instead only providing statistical controls over data obtained during the course of a single run with the goal of determining the length of time the simulation run should last. The Akaroa2 API was used to develop a system to launch multiple independent runs for each scenario, collecting and tracking the final statistics calculated at the end of each run. It is possible to define multiple statistics to collect from each simulation and ensure each scenario is run just enough times in order for all of the gathered statistics to fall within their statistical goals. By only running as many times as necessary for the specified goals, very little processor time is wasted from guessing how many runs are necessary for the desired confidence. The developed framework is currently capable of deploying multiple runs in parallel on the same machine, however it does not take advantage of Akaroa2's features to deploy and manage multiple runs across different machines.

For the following evaluations, the number of message copies generated, the percentage of messages that were delivered, the latency of delivered messages, the number of dropped messages, as well as the number of messages cured were all tracked using the Akaroa2 framework. The number of message copies generated had the highest statistical goal, requiring a relative error of $\pm 5\%$ with 95% confidence. Delivery probability was required to achieve an error of $\pm 7.5\%$ with 95% confidence. The remaining features were only required to achieve a high error rate of $\pm 10\%$ with a low 90% confidence. This allowed the Akaroa2 framework to calculate and track the error rates for these statistics as well, but it ensured that the number of simulation runs was not increased by the collection of these statistics. It was thought that sufficient consistency with the number of copies generated and percentage of messages delivered would be enough. Finally, a maximum of 25 runs was placed on each scenario, such that the results were taken as is once this limit was reached,

even if all of the statistical goals had not yet been reached. Only a small handful of results required all 25 simulation runs and did not fully meet their statistical goals.

5.3 Mobility Models

The Levy Walk and Village mobility models were used for the routing algorithm evaluations. The Levy Walk model was used to present results for randomized mobility containing no underlying pattern, while the Village model introduces underlying mobility patterns that could be utilized by mobility aware routing algorithms such as PRoPHET, Binary Spray and Focus, and Estimation based Erasure Coding. An in-depth description of these models follows.

5.3.1 Truncated Levy Walk Mobility

The Truncated Levy Walk model is a purely statistical model that draws values from a random distribution to determine the distance traveled and angle of movement for each new destination, as well as the pause time between movements. It uses a power law distribution $p(l) \sim \frac{1}{l^{1+\alpha}}$ [38] with $0 < \alpha \leq 2$ to generate the flight lengths, which is a heavy-tailed distribution causing a majority of the flight lengths to be short but with occasional longer flights as well. Angles of movement are pulled from a uniform distribution. Truncated Levy Walk is a type of power law distribution that has been studied extensively for animal patterns and recently has been shown to be promising as a model for human mobility [22, 38]. Our implementation is directly based on the model by Rhee et al. [38].

The primary characteristics of the Levy Walk mobility model can be changed by modifying the power law exponent, α, and the truncation factor, τ. The α parameter changes the ratio of short flights to long flights, with lower values of α causing an increased percentage of longer flights. Modifying the ratio of flight lengths greatly impacts the mobility pattern, as an increased number of long flights

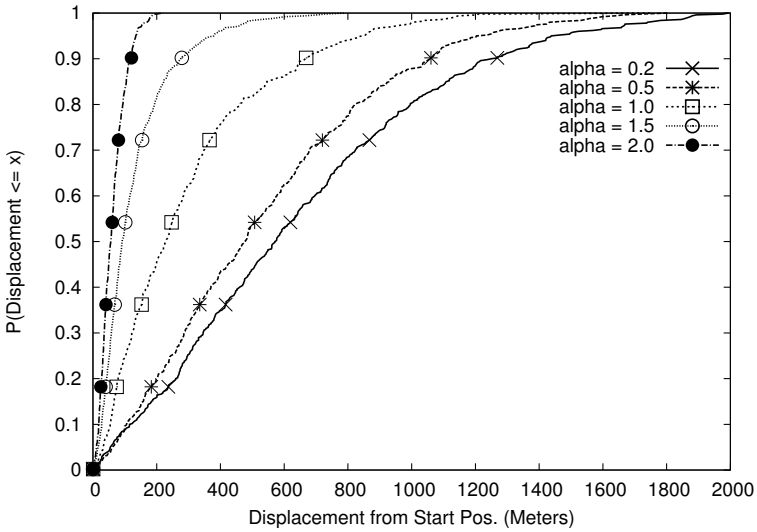
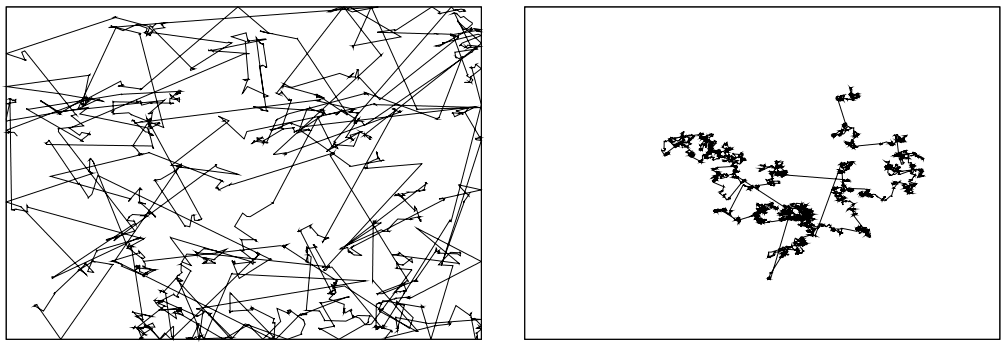


Figure 5.1: Displacement CDF on a 2km x 2km grid over 10 minutes with flight truncation = 1km, pause truncation = 1000s, and pause $\alpha = .5$

allows nodes to travel further in the same period of time. Figure 5.1 shows the cumulative distribution function of the likelihood a host will be a particular distance from its starting position at the end of the simulation run. It shows that in this scenario, $\alpha = 2.0$ ensured a host was no further than 200 meters from its start position after 10 minutes of simulation, whereas $\alpha = 0.2$ resulted in some hosts travelling as far as 2000 meters from their start position over the same period of time. Figure 5.2 shows the mobility of a single host under the same simulation parameters used to generate the CDF in Figure 5.1, while differing the value of α. As can easily be seen, $\alpha = 1.5$ results in a much more localized pattern than $\alpha = 0.5$. The truncation factor, τ prevents flight lengths and pause times above a threshold value. Intuitively, this represents the real-world limitations imposed by our environment, such as obstacles limiting flight length and the inability and unlikelihood of remaining still for extremely long periods of time. The simulation results presented in this section used an α value of 0.5 for both the flight lengths and pause times. A truncation factor of 2500 meters was used for the flight length,



(a) α=0.5 (b) α=1.5

Figure 5.2: Motion traces for and Levy Walk with varying values of α

as well as a maximum pause times of 1000 seconds.

5.3.2 Village Mobility

The Village mobility model consists of a set number of villages and the villagers who inhabit them. The villages are dispersed throughout the simulation landscape, with the villages below a threshold distance from each other being connected by roads. These roads create a transportation network by which villagers can travel between villages. Figure 5.3 shows the arrangement of villages used for these evaluations and some example movement of villagers between them. The circle surrounding each village represents its area, with the villagers currently within that village only allowed to move within this circle.

During initialization, each villager is randomly assigned and placed within a home village that it remembers throughout the course of the simulation. For these evaluations, the two largest villages were assigned 15 villagers each, with the four remaining smaller villages assigned five villagers each. The number of villagers assigned to a particular village is used to determine how popular each village is, which is then used to determine how likely that village is to be chosen as a destination by other villagers.

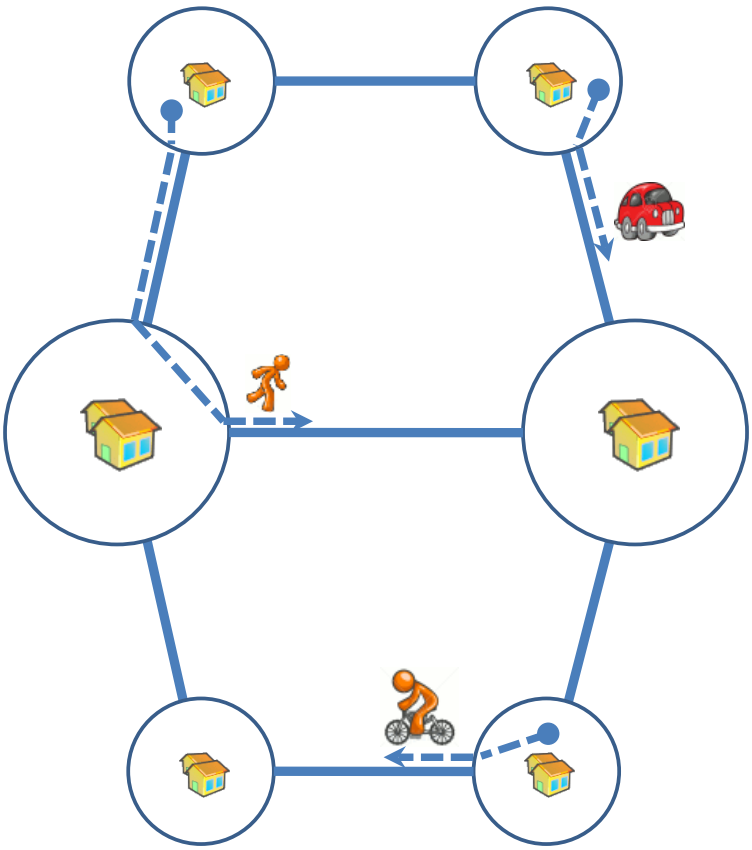


Figure 5.3: Village Mobility

The mobility pattern of each host is determined by a series of destination choices. When determining its next destination, a host can choose to stay in its current village or choose to visit another one. When a host chooses to stay within its current village, it randomly picks another location within the village's circular area to move to and upon reaching that location will again decide whether to remain in the village or move to another one. Simulation parameters are used to determine the probability of making each decision. If a host is currently within its home village, 95% it will choose to remain there. If it chooses not to remain in its home village,

it randomly chooses another village, with each village's probability of being chosen being proportional to its population size. Once away from its home village, a host will choose to return home 15% of time, with the other 85% of the time evenly split between staying in its current non-home village, or moving to another non-home village. These movement probabilities are meant to establish a pattern of mobility among hosts, such that hosts have higher delivery likelihoods for hosts that are assigned to their same village. While a host is away from its home village, it should be a desirable choice to deliver any messages destined to hosts that also reside in same village.

When moving to a different village, Dijkstra's shortest path algorithm is used to determine the path of roads and villages that are traversed to reach the final destination. A move to a new village comprises a number of separate movements, as the host must first move to the road entrance at the edge of the village that connects the villages together. From there, it moves straight along the road to the next village, and repeats the process if additional hops through villages are necessary. All movement within a village is done by walking at a set walking speed, but hosts travelling between villages can also use bicycles or cars capable of traveling at higher speeds. The likelihood of choosing each method of transportation is proportional to the distance between the starting point and the destination. The probability of choosing a car, $P(car)$, is the ratio between the distance to the next village and the maximum possible distance between villages, such that a car is more likely to be used over longer distances. When a car is not used, a 2:1 ratio is used between biking and walking between villages.

In principle this mobility model is similar to the "Community Model" proposed by Lindgren et al. [32]. It is enhanced in that the mobility between villages is explicitly modeled, which was not done for the "Community Model". The scenario used for these results also did not include a stationary gateway node at each village,

instead requiring all transfer of messages to occur between mobile hosts.

5.4 The Impact of Copy Limits

The first architectural feature evaluated was the impact of the copy limit restriction on algorithmic performance. The cure exchange feature was enabled for all of the results obtained in this section. The Flooding, Epidemic, PRoPHET, Binary Spray and Wait, as well as Binary Spray and Focus algorithms were evaluated with copy limit values in the range $[1, 50]$. Instead of using the same range of copy limit values, the erasure coded algorithms were instead evaluated using the range $[1, 8]$. Unlike other algorithms, which can infinitely duplicate messages after they have been created, the erasure coding process creates all of its replicated coded blocks when a message is first received by the application layer. Therefore with each host creating six messages, a maximum copy limit (or replication factor in erasure coding terms) of eight results in 48 slots of the erasure coded buffer being filled purely by the messages each host generates themselves.

Figure 5.4 shows the number of message copies generated by each algorithm as the copy limit parameter was increased. The Flooding, Epidemic, and PRoPHET algorithms all implement a localized copy limit restriction, which does not impose a hard cap on the number of copies of any one message that may be generated. The lack of a hard restriction for these algorithms accounts for the much steeper increase in the message copies plot compared to the other algorithms that impose a globally enforced restriction on message copies. The algorithms with global copy limit schemes show a linear increase in their message copy plots as the copy limit parameter increases. A copy limit value of 50 is near the maximum number of copies that could possibly be made of a single message for these simulations as this allows a copy of a message to be created for every host in the network. A message could only have have more than 50 copies if it was first dropped by a host, and then that

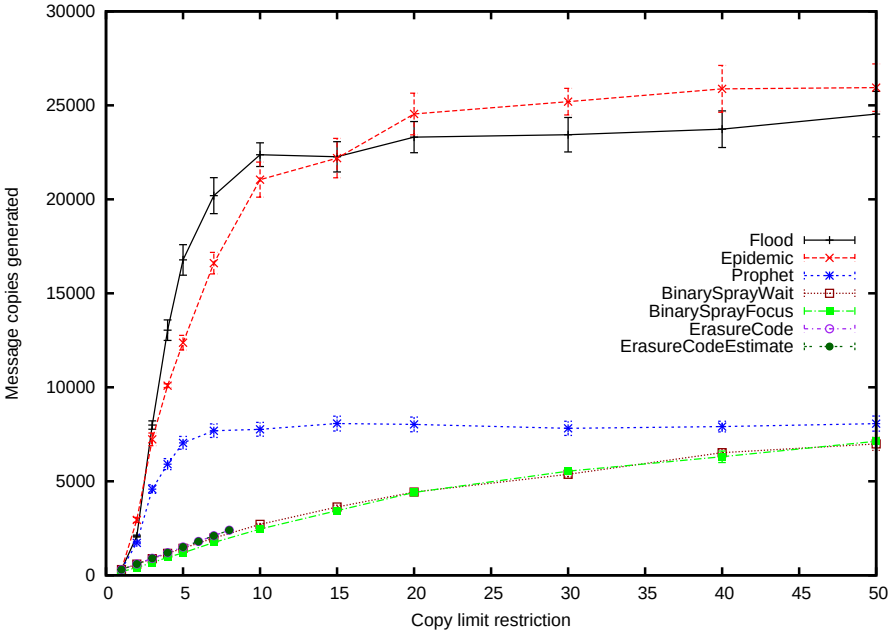


Figure 5.4: Impact of the copy limit parameter on number of generated message copies. The Levy Walk mobility model was used.

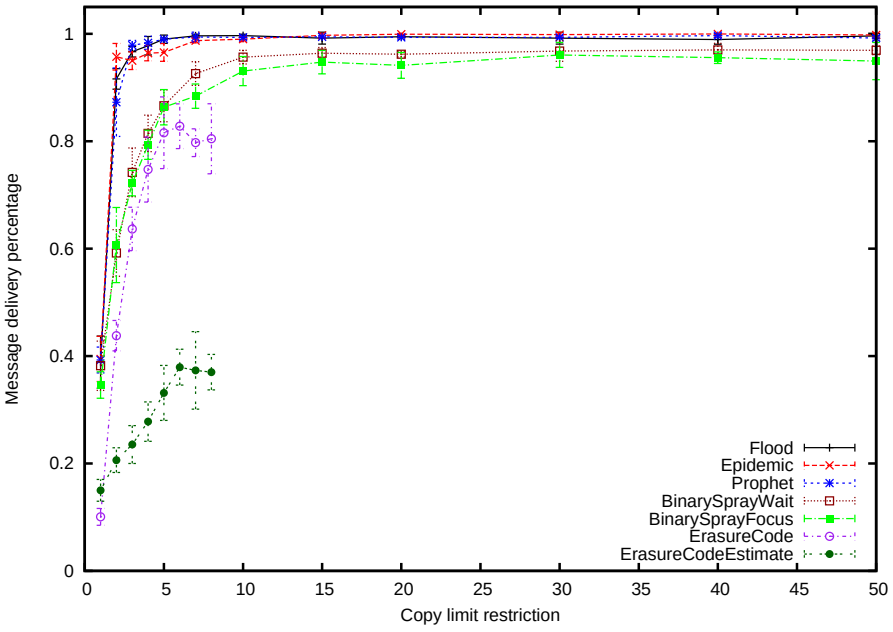


Figure 5.5: Impact of the copy limit parameter on message delivery probability. The Levy Walk mobility model was used.

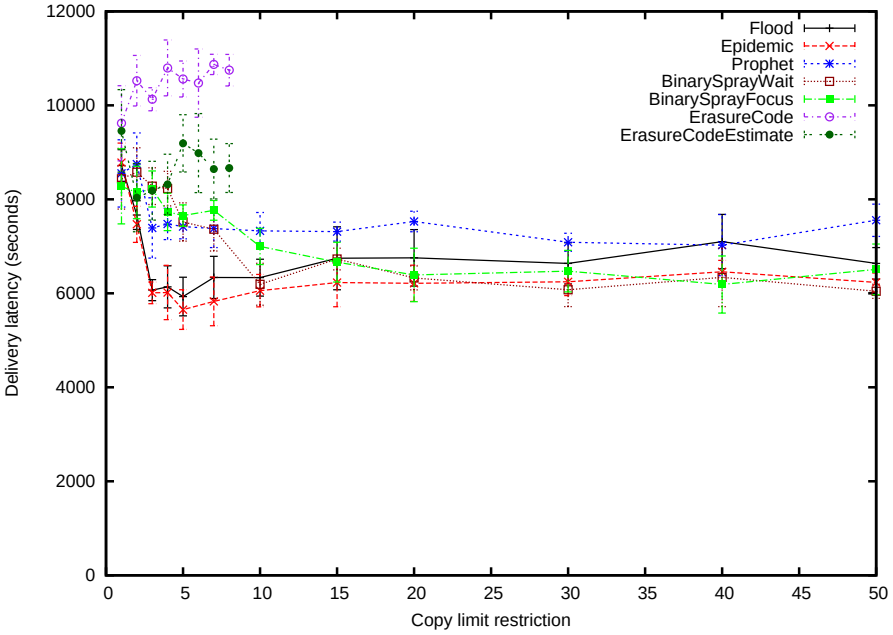


Figure 5.6: Impact of the copy limit parameter on message delivery latency. The Levy Walk mobility model was used.

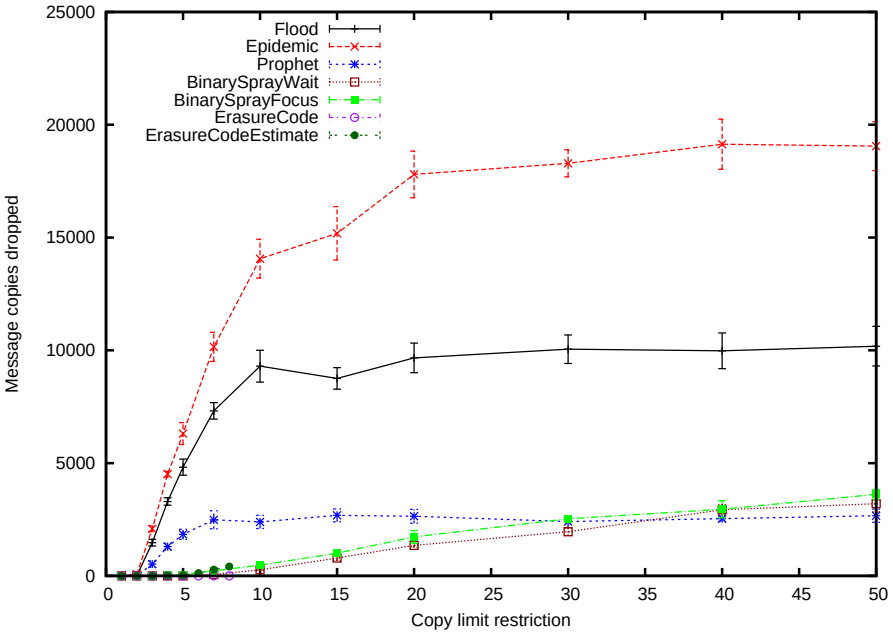


Figure 5.7: Impact of the copy limit parameter on number of messages dropped. The Levy Walk mobility model was used.

host received another copy of the same message later on while encountering another host.

Figure 5.4 shows a very unexpected result in that the Epidemic algorithm generated more message copies than Flooding for copy limit values greater than 15. This is not intuitive since the Epidemic algorithm is meant to be an improved version of Flooding that utilizes summary information to not send a message copy to another host when it already has that message in its buffer. While lacking a summary exchange feature, the Flooding algorithm does locally keep track of which messages it has already delivered to which hosts so that it will not deliver the same message to a host more than once if they continue to encounter one another. Interestingly enough, the summary exchange implementation used for all of the other algorithms does not ensure that the same message is not sent multiple times to the same destination. All of the non-Flooding algorithms solely rely on the summary information received by the other host to determine whether or not to send a message copy, not taking into account whether they had previously sent the message to that host. What is occurring in the results shown in Figure 5.4 is that the limited buffer space is causing hosts to receive message copies while their buffers are full, which subsequently forces their lowest priority message to be dropped. With Flooding if, for example, host A delivered a message to host B, but host B was forced to drop said message, host A would not be allowed to send that same message to host B again if they were to encounter one another later on. With all of the other algorithms, host A would send another copy of the message when it encountered host B again. We originally thought that the summary information provided by the other host would be sufficient, however these results show that under certain circumstances there is a significant benefit in locally tracking which messages have been delivered to each host. Such a feature can certainly be added to the summary exchange used by the other algorithms, and these results indicate

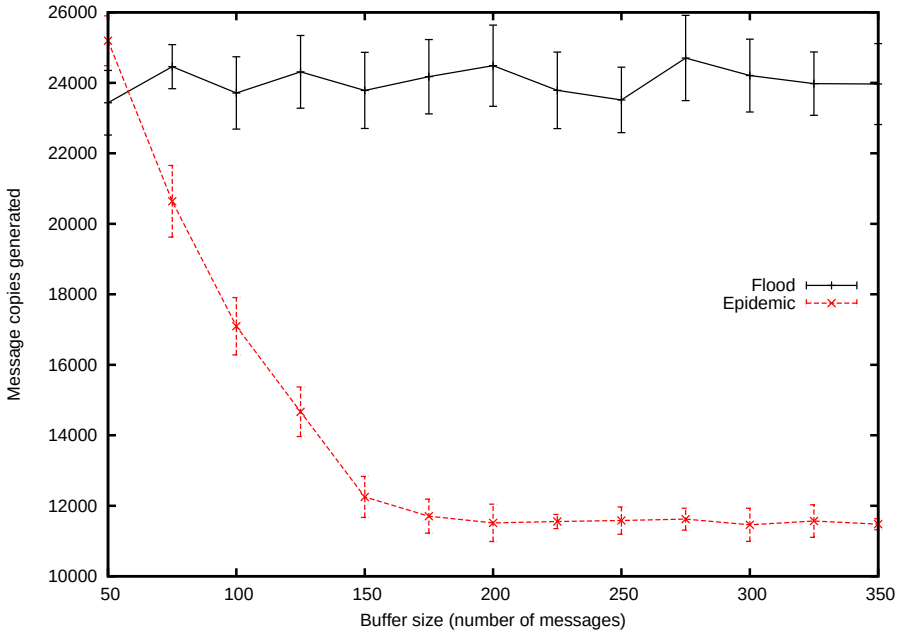


Figure 5.8: As the buffer size increases, the performance advantage of local message delivery tracking diminishes for Flooding, while Epidemic performance continues to improve. The Levy Walk mobility model was used.

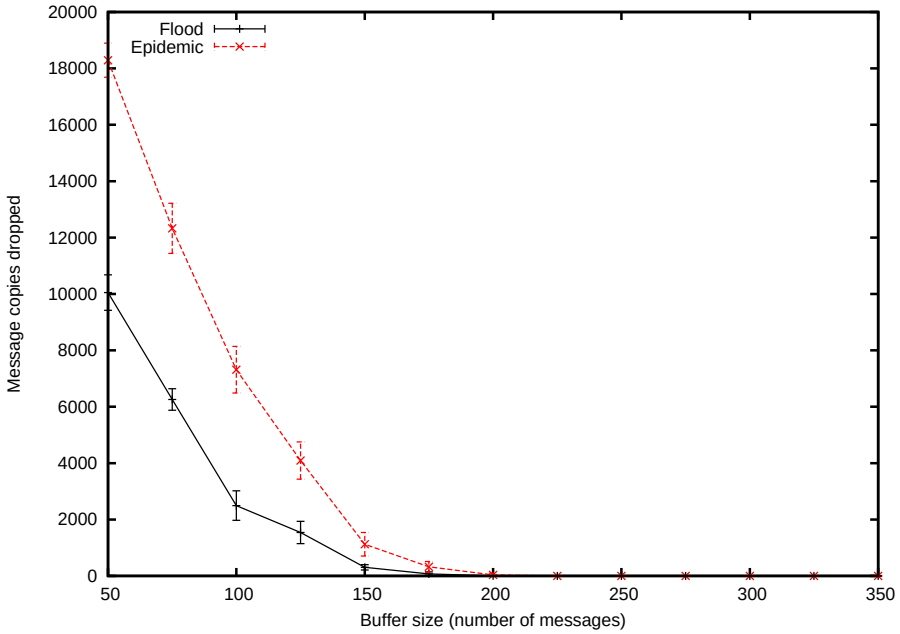


Figure 5.9: The number of messages dropped by Flooding and Epidemic as the buffer size increases. The Levy Walk mobility model was used.

it would provide a significant performance benefit under these simulation scenarios. Figure 5.8 shows how the number of message copies generated changes as the buffer size is increased for both Flooding and Epidemic. As the buffer size increases, the efficiency of Epidemic increases while Flooding stays roughly the same, showing that the advantage of locally tracking message deliveries diminishes as the number of dropped messages decreases (Figure 5.9).

Even though Figure 5.4 shows results for the non-predictable Levy Walk mobility model, the PRoPHET algorithm still generated a vastly lower number of message copies than Epidemic while maintaining a similar delivery probability. The PRoPHET algorithm determines which messages to forward based on its mobility prediction data, and even though this prediction information is not reliable given the underlying mobility model, PRoPHET still makes an additional decision not to forward some messages that Epidemic does not make. This extra forwarding decision, whether accurate or not, accounts for the difference in message copies between the two algorithms. Further explanation of the impact of mobility is discussed in Section 5.5.

The number of messages dropped by each algorithm as the copy limit increases is shown in Figure 5.7. As expected, with Flooding and Epidemic generating many more message copies than the other algorithms, they necessarily dropped many more copies as well. Almost no message copies were dropped in any of the globally restricted schemes for copy limit values of 10 and under. Figure 5.6 shows how the latency of the delivered messages changed as the copy limit was varied. The PRoPHET algorithm shows a consistently higher latency than Epidemic, which is a consequence of being more discriminant and not taking advantage of as many forwarding opportunities as Epidemic does. The erasure coded algorithms both show higher latencies than any of the other algorithms, which is also expected as they both rely on a minimum set of relays to reach the destination instead of allowing a

single advantageous encounter to deliver an entire message.

The Flooding, Epidemic, and PRoPHET algorithms performed the best in terms of delivery probability, consistently delivering 98% or more messages with a copy limit value greater than three. Even with a copy restriction of 50, the Binary Spray and Wait and Focus algorithms do not perform quite as well, never getting above 96% delivery during the course of the simulation. This is the tradeoff of generating roughly 7000 message copies as opposed to around 26000 messages for a copy limit value of 50. Although this greatly increased number of message copies did not significantly degrade the effectiveness of Flooding and Epidemic to deliver messages, this result will likely not hold up with either smaller buffers or more network traffic. The simulation scenario used was not strenuous enough to demonstrate the performance degradation that should occur with their much higher rate of generating message copies.

The erasure coded algorithms did not perform particularly well compared to the other protocols with the chosen simulation scenarios. The Estimation Based Erasure Coding algorithm was at a significant disadvantage, which could have been caused by the use of predicted mobility patterns for forwarding decisions in a network with no predictable mobility patterns. One explanation for the performance difference between Estimation Based Erasure Coding and Original Erasure Coding is the number of relays that were utilized by each approach. Both erasure coding algorithms generate the same number of coded blocks per message (6*copyLimit), however, the Original Erasure Coding algorithm distributes two of its coded blocks to each unique relay it encounters. This requires a minimum of three relays to encounter the destination before a message can be delivered, which accounts for its much higher latency values (Figure 5.6). The Estimation Based Erasure Coding algorithm instead distributes a proportion of its coded blocks to each relay it encounters based on its estimated delivery probability, while also requiring a host to

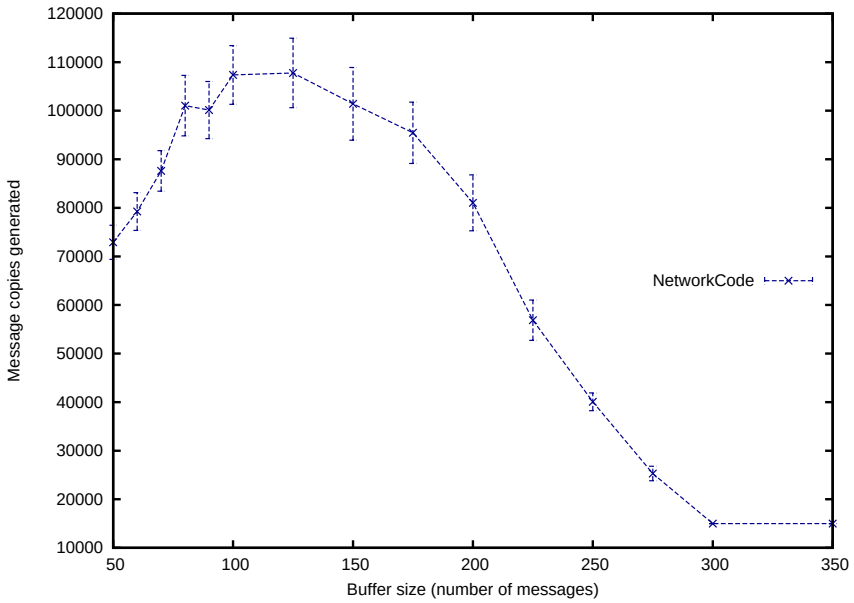


Figure 5.10: Network coding generates an extraordinarily large number of message copies in limited buffer scenarios. These results were obtained using the Levy Walk mobility model.

stop spreading coded blocks once six or fewer blocks remain in its buffer. What occurs is that the hosts often end up with three, four, or five coded blocks in their buffer for a particular message. This is still not enough to entirely decode the message, thereby still requiring a minimum of two hosts to encounter the destination for the message to be delivered. Additionally, the fact that the majority of the hosts end up with three, four, or five coded blocks implies that fewer unique hosts will be carrying blocks of each message as compared to the Original Erasure Coding algorithm, which only allows two blocks per host. The requirement to stop spreading blocks once six or fewer blocks remain has the effect of reducing the number of relays a message is spread to, and as is seen in the results, has a significant impact on its performance.

The Network Coding algorithm exhibited some very interesting and surprising behavior during evaluation. It was discovered that the performance of network coding was extremely poor when the buffer size was small enough to require mes-

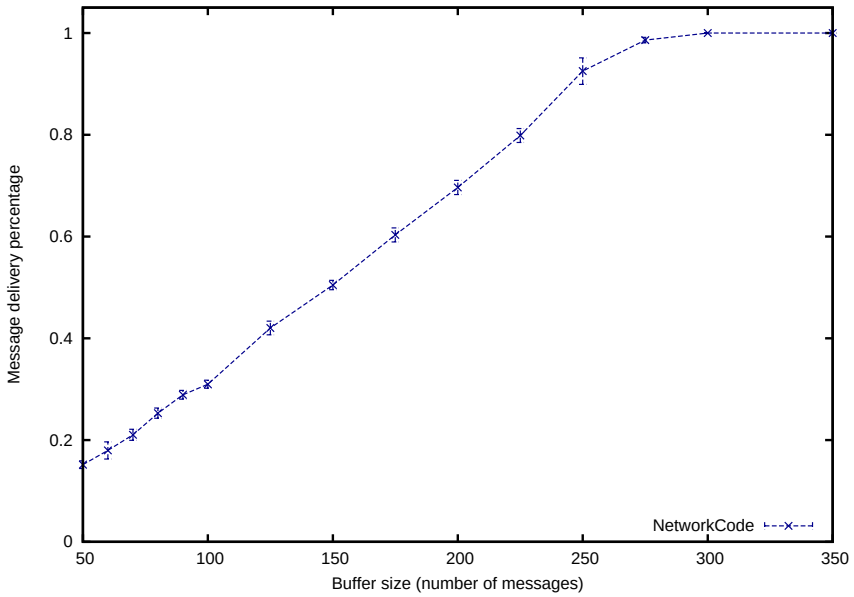


Figure 5.11: All messages were delivered when buffer space no longer a limited resource.

sages to be dropped from the network. Figures 5.10 and 5.11 show the number of copies network coding generated as well as its delivery probability as the buffer size increases. A copy limit restriction of 50 was used for these results. There is a veritable explosion in message copies with network coding compared to the other algorithms. For the same buffer size of 50, Network Coding generates roughly 72000 copies, while the next worse performing algorithm, Epidemic, only created 26000 copies with the same parameters. Although initially thought to be a bug, this is actually a consequence of how message summary information is exchanged and is related to why Flooding outperformed Epidemic in some cases. With the current Network Coding implementation, two hosts exchange their decoding matrices during summary exchange. With the other host's decoding matrix, each host can check to see whether a randomly coded message from its own buffer will increase the rank of the destination's decoding matrix. As shown in Figures 5.10 and 5.11, this approach works perfectly well when buffer space is not limited (i.e., the buffer size is ≥ 300). However, consider what occurs when buffer space becomes limited. A host receives

the summary information and determines that it should generate a coded message for the other host as the coded message will increase the rank of the other host's decoding matrix. The destination's buffer, however, is currently full so upon receiving the message the receiving host removes a message from its buffer. The host that sent the message has no mechanism to know that this occurred. Therefore, if these two hosts were to meet again, which is a common enough occurrence, the exact same exchange would very likely occur, with one host sending copies to the other that were dropped. With the other protocols, each message in a buffer has a unique identifier that can be exchanged in the summary information, which allows a host to not send a message to another host if it currently has the message in its buffer. Network Coding has no such mechanism and instead must exchange abstract information about coefficient matrices that are not as specific or efficient. This inefficiency is compounded by the fact that messages can not be decoded and delivered until the entire generation can be decoded. What often occurs is that many hosts do not receive enough coded packets to decode an entire generation, resulting in buffers that are full of partially decoded messages that cannot be delivered to any applications. There are clearly some issues that need to be further analyzed and understood with the Network Coding algorithm. It has yet to be determined whether these issues are fundamental flaws or simply whether alternate strategies could be employed to mitigate these inefficiencies. The further analysis of these issues is left as future work.

5.5 The Impact of Mobility Prediction

The performance of the routing algorithms was also evaluated using the Village mobility model. Figures 5.12 - 5.15 show the number of copies generated, delivery probability, delivery latency, and number of dropped messages respectively under the same simulation scenario used in Section 5.4, with the only difference being the

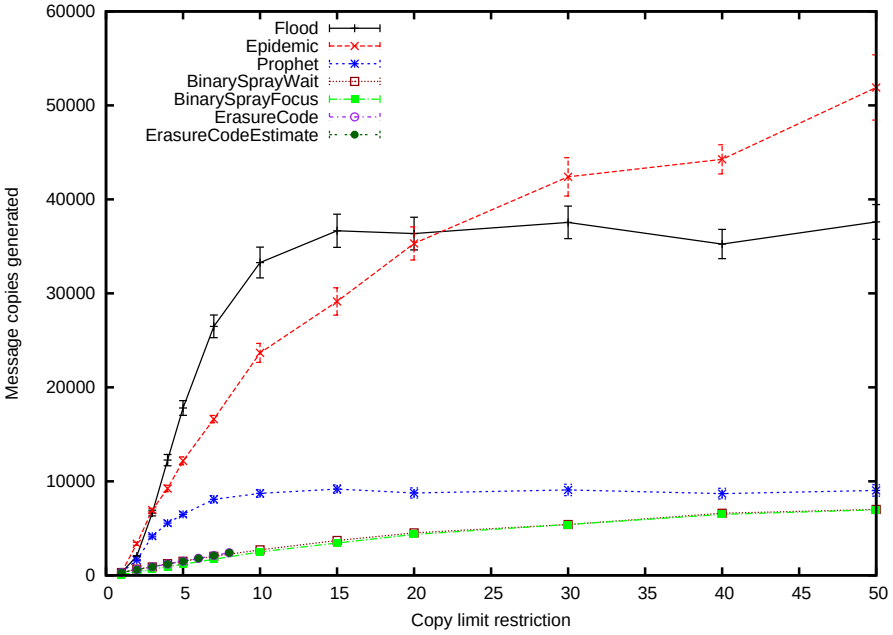


Figure 5.12: Impact of the copy limit parameter on number of generated message copies. The Village mobility model was used.

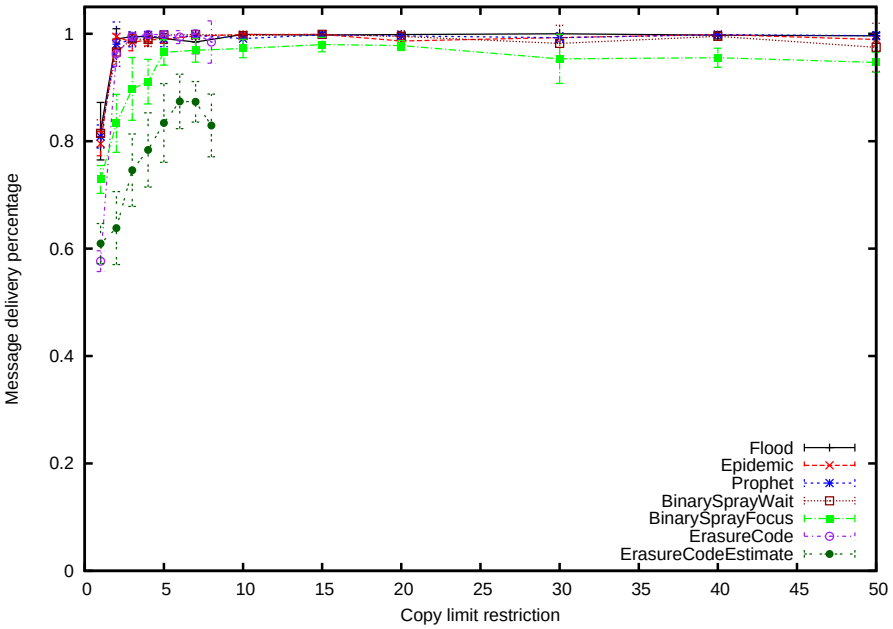


Figure 5.13: Impact of the copy limit parameter on message delivery probability. The Village mobility model was used.

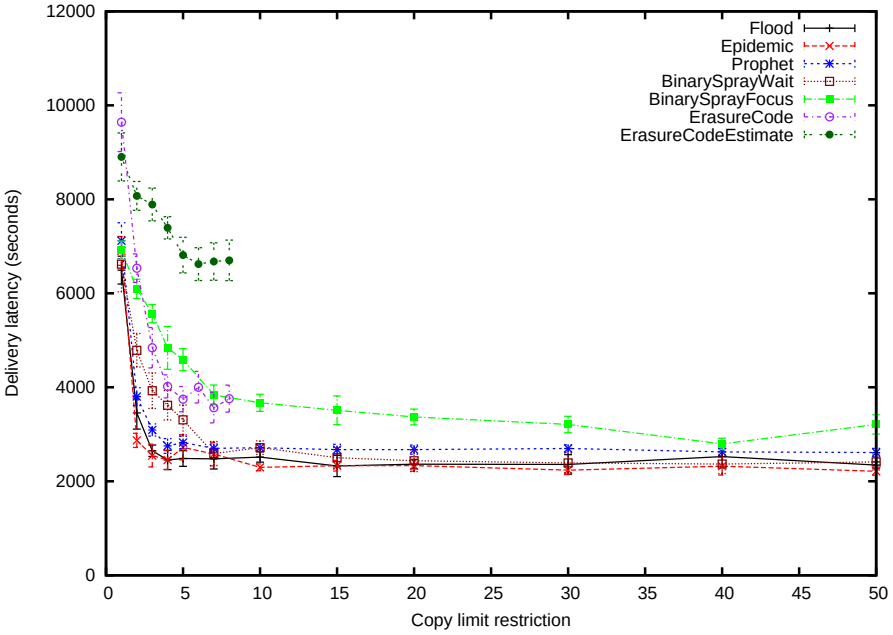


Figure 5.14: Impact of the copy limit parameter on message delivery latency. The Village mobility model was used.

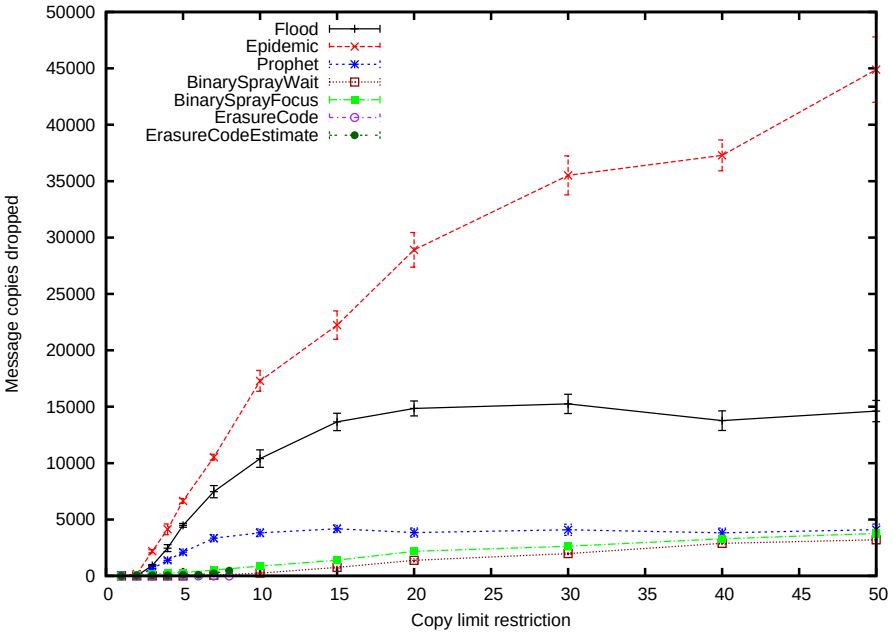


Figure 5.15: Impact of the copy limit parameter on number of messages dropped. The Village mobility model was used.

mobility model. Overall, the relative number of copies generated by each protocol is similar to the Levy Walk scenario. The delivery percentages are slightly different, with a significant increase being shown for the erasure coded algorithms. This is likely caused by the greater number of contacts available with this mobility model as mobility is restricted to a much smaller number of locations. The delivery latency is much lower across the board, as would be expected with this mobility model since there will be more frequent contacts with other hosts. The Binary Spray and Focus algorithm shows a longer average latency than the Binary Spray and Wait algorithm, which is unexpected. It would be expected for Binary Spray and Focus to outperform Binary Spray and Wait as it utilizes mobility prediction to forward a message toward its destination after all of the messages copies have been spread. The performance degradation is not due to the mobility prediction, but instead it is with the transfer of message ownership during the forwarding process. Although acknowledgements are utilized during the exchange, these acknowledgments only indicate that the other host received the message, not that it was successfully inserted into its buffer. Without acknowledgement that the message was actually stored in the other host's buffer, a pure forward of a message from one host to another is at risk of dropping the message entirely if the destination host does not have the buffer space to store it. Therefore, the pure forwarding of messages during the "focus" phase of the Binary Spray and Focus algorithm places message copies at higher risk and results in a reduction in performance. The Estimation Based Erasure Coding algorithm is also affected by the unreliable transfer of ownership during its pure forwarding phase. As the results show, the Estimation Based Erasure Coding algorithm still performs poorly compared to the other algorithms, even in a simulation with a more predictable mobility pattern. Quite simply, this algorithm does not perform well compared to the other algorithms with its current implementation.

In order to further evaluate whether the mobility prediction was having a

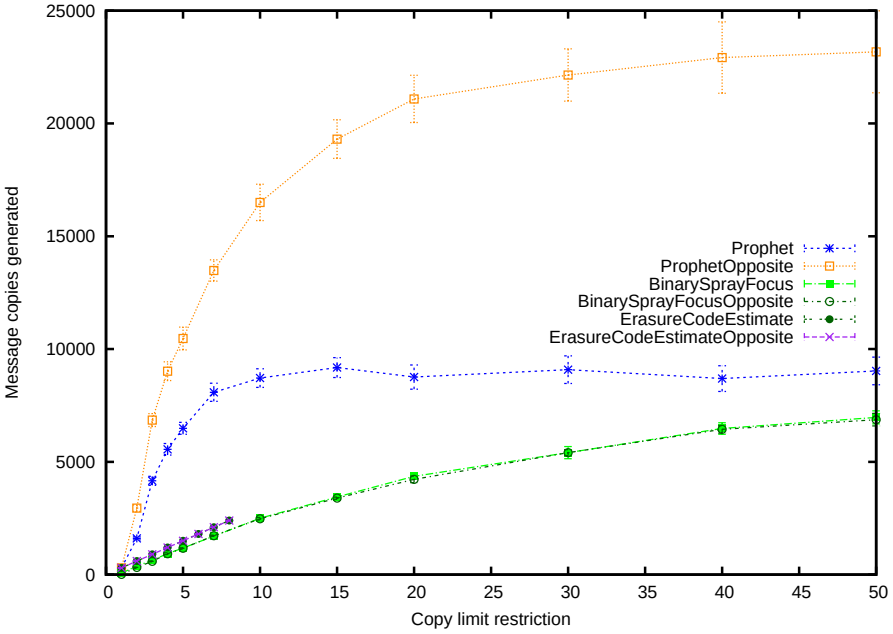


Figure 5.16: Comparison of the number of message copies generated between the mobility predictive algorithms and their "Opposite" implementations. The Village mobility model was used.

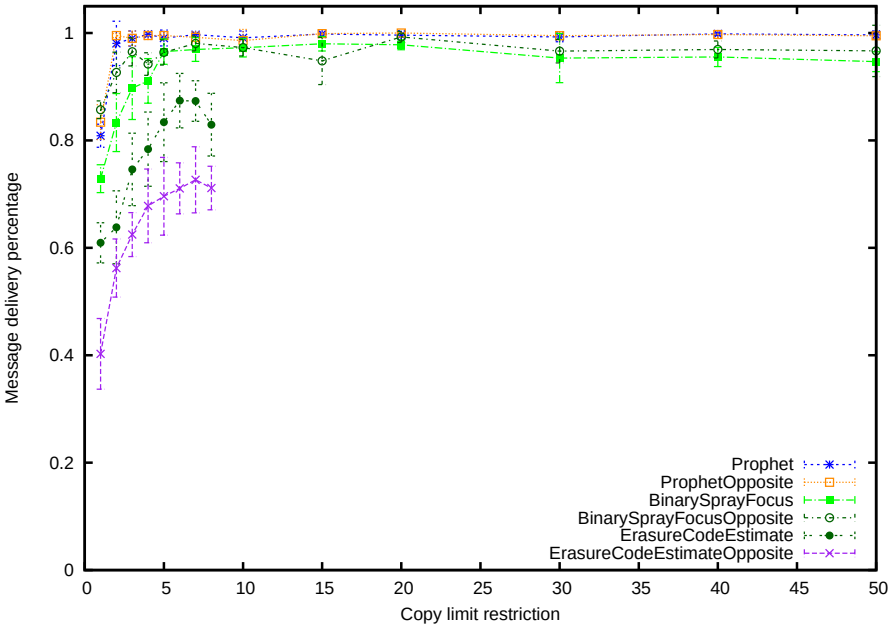


Figure 5.17: Comparison of delivery probability between the mobility predictive algorithms and their "Opposite" implementations. The Village mobility model was used.

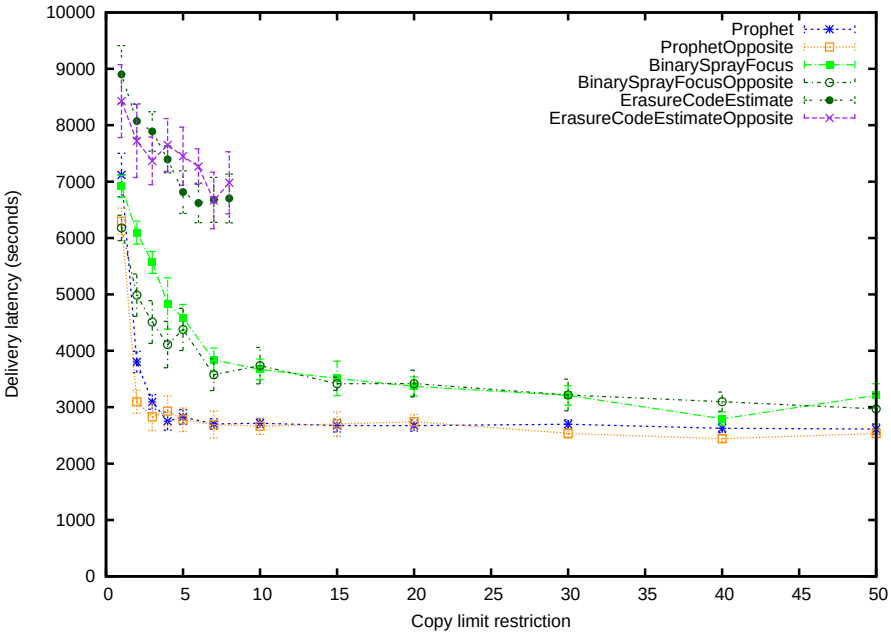


Figure 5.18: Comparison of delivery latency between the mobility predictive algorithms and their "Opposite" implementations. The Village mobility model was used.

positive impact, the PRoPHET, Binary Spray and Focus, and Estimation Based Erasure Coding algorithms were all compared against slightly modified versions of their algorithms. An "Opposite" version of each algorithm was created, where any time a decision was made based on mobility prediction information, the "Opposite" algorithm would make the exact opposite decision the original protocol would with that information. If the mobility prediction is useful, then there should be a noticeable difference in performance between the two versions. Figures 5.16 - 5.18 show the results comparing the original and opposite versions of the algorithms. With the PRoPHET algorithm, there was a significant difference in the number of copies that were generated, however the delivery probabilities remained virtually the same. The improvement in message copy generation over the "Opposite" version does indicate that the mobility prediction data was put to good use by the PRoPHET algorithm. There was also a significant difference in delivery percentage between the original

and opposite versions of the Estimation Based Erasure Coding algorithm. In this case both versions of the algorithm are generating the exact same number of copies, so the performance increase can only be attributed to the mobility prediction algorithm. This does show that there is a predictable nature to the Village mobility model being used and that the Estimation Based Erasure Coding algorithm was able to take advantage of it.

The results for mobility prediction do show some performance improvements when it is utilized, however it also shows it to only be advantageous under certain circumstances. There are two components important to the performance of an algorithm utilizing mobility prediction: the mobility prediction mechanism itself and how that mobility information is used to make decisions when forwarding messages. For all of the algorithms presented, the same mobility prediction mechanism from the PRoPHET paper [32] was used. Additional mobility prediction schemes from the literature should be explored to determine whether or not they can improve the performance of these algorithms. The simulation architecture makes it very straightforward to modify and replace the prediction mechanism without modifying any of the other algorithmic features. It would also be interesting to determine under what circumstances mobility becomes predictable enough to be useful, as well as further exploring the significance of its impact over other architectural features such as copy and hop limit restrictions. The exploration of these questions is left for future work.

5.6 The Impact of Cure Exchange

As was described in Section 3.2, the cure exchange feature will almost always have a beneficial effect on the performance of a DTN routing algorithm as purging delivered messages from buffers only serves to create more space for as-yet undelivered messages. As expected, a noticeable performance advantage was achieved with this

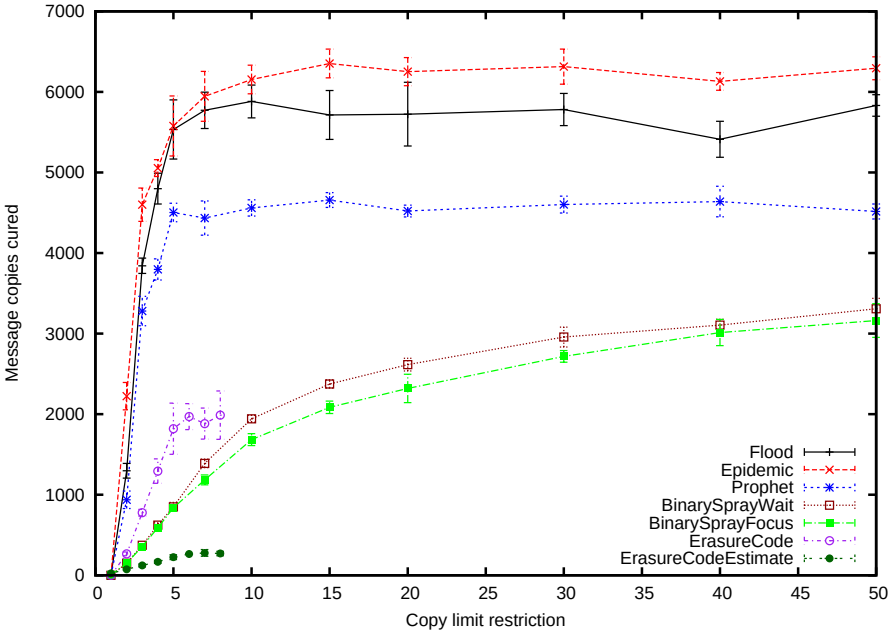


Figure 5.19: Number of messages removed from hosts' buffers using the cure feature. The Levy Walk mobility model was used.

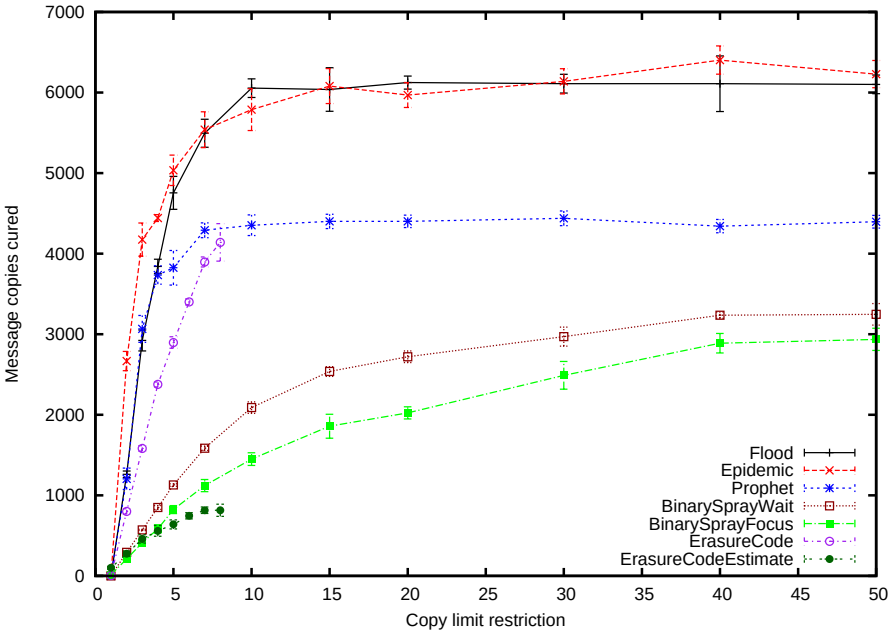


Figure 5.20: Number of messages removed from hosts' buffers using the cure feature. The Village mobility model was used.

feature for the evaluations that were performed. Figures 5.19 and 5.20 show the number of messages that were cured by each algorithm as the copy limit variable is increased. As expected, the algorithms that produced the most number of message copies also cured the most messages. This intuitively makes sense, as more message copies implies a message was present in more buffers throughout the network, providing more opportunities for those messages to be cured once they had been delivered to their destinations. Figures 5.21 - 5.26 show the number of copies generated, delivery probability, and number of messages dropped for both mobility models when the cure feature was disabled. Comparing these results to those in Figures 5.4 - 5.7 and 5.12 through 5.15 shows that the delivery performance was considerably improved by the cure feature. Without curing, none of the algorithms managed to deliver all of their messages, with approximately a 5% drop in delivery rate for all of the algorithms. The number of copies generated also roughly doubled. Under the scenarios evaluated, the results are quite significant and indicate this feature should be seriously considered for any system.

Some additional insight was also gained into the effectiveness of the cure feature for the Network Coding algorithm. With the cure feature, there are two ways that a message first gets added to the cure lists that are exchanged between hosts. The first method is when a message's destination host adds that message's ID to its own cure list after delivering it to its application layer. Secondly, if a host delivers a message to its destination and receives an acknowledgment, it can safely assume that the message was delivered and add it to its own cure list. Network Coding is not capable of taking advantage of the second method as the messages being sent between nodes do not have a single destination; instead they are combinations of multiple packets with multiple destinations. This implies that the cure information is propagated more slowly throughout the network as instead of having two hosts add the message to their cure lists upon delivery, only the actual destination does

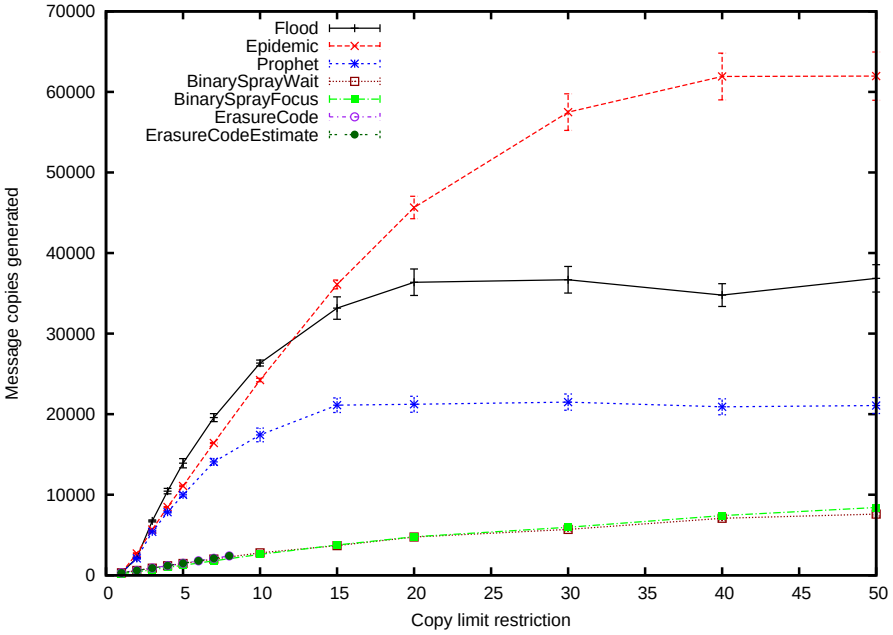


Figure 5.21: Number of message copies generated without the cure feature, using the Levy Walk mobility model.

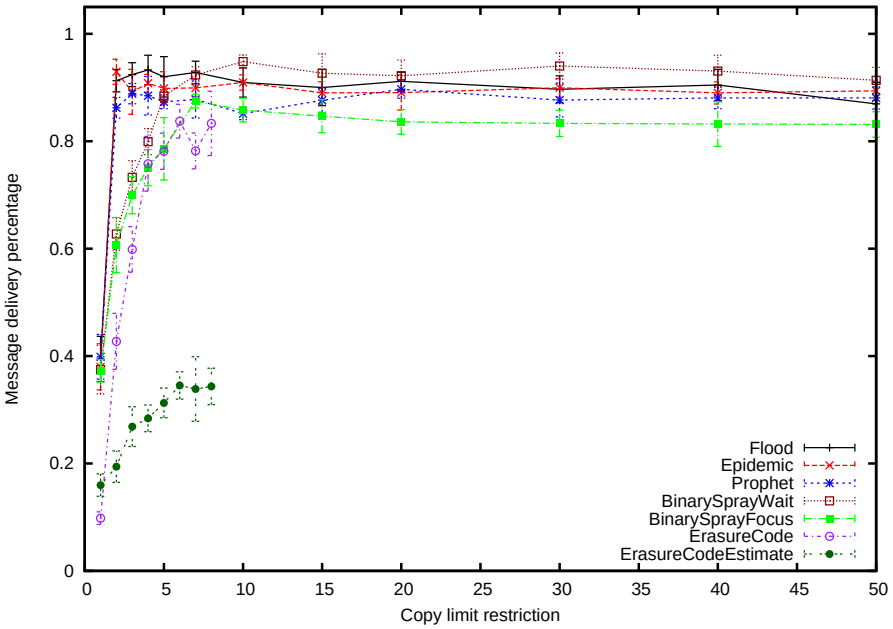


Figure 5.22: Message delivery probability without the cure feature, using the Levy Walk mobility model.

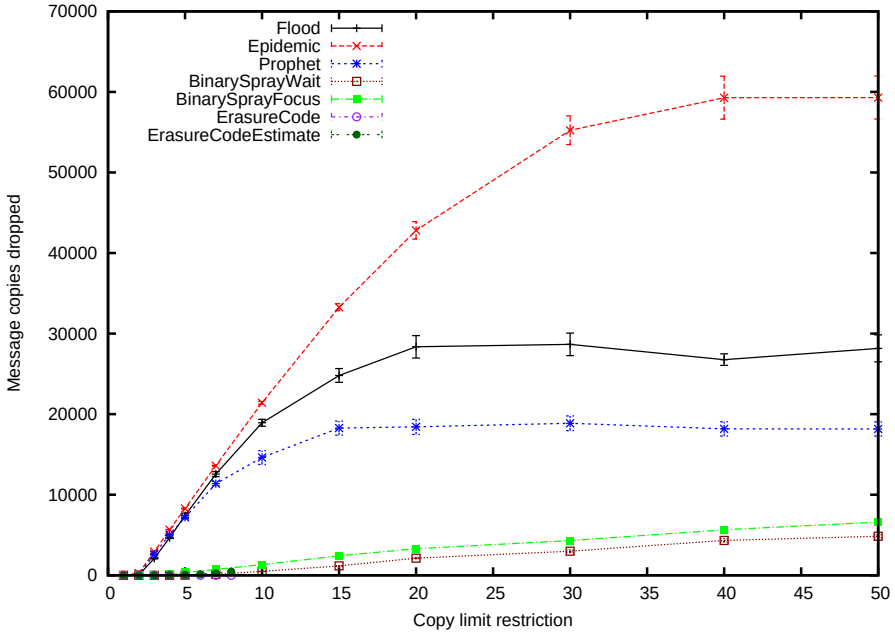


Figure 5.23: Number of message copies dropped without the cure feature, using the Levy Walk mobility model.

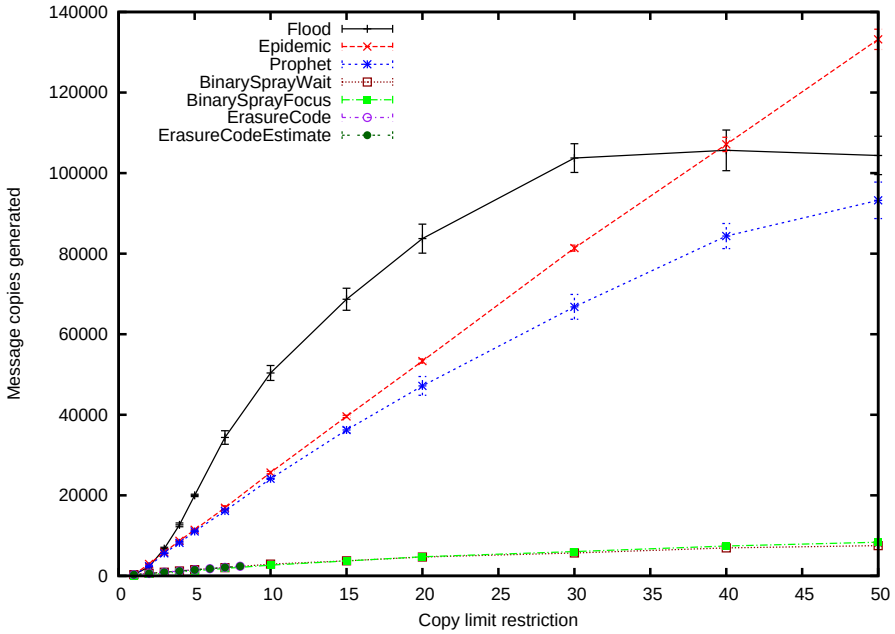


Figure 5.24: Number of message copies generated without the cure feature, using the Village mobility model.

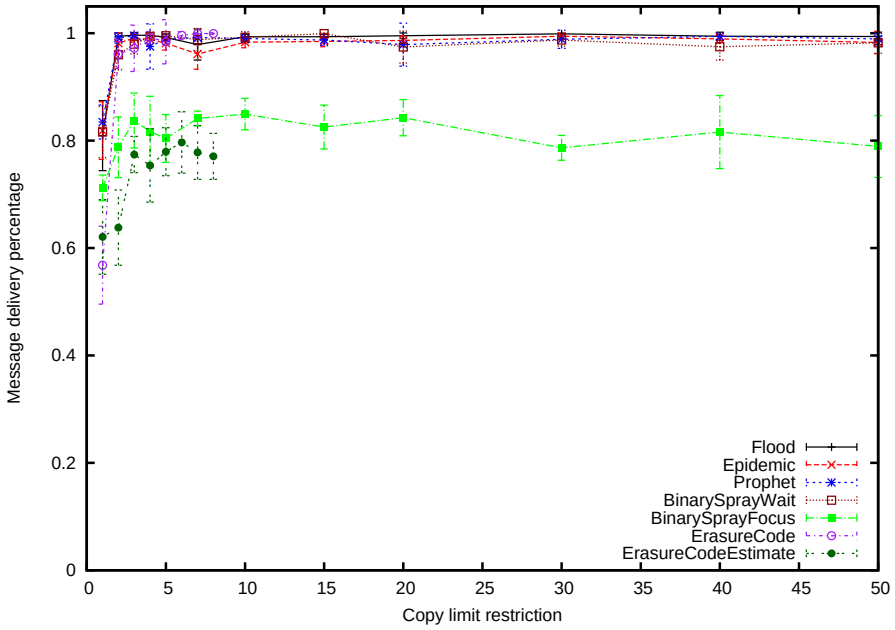


Figure 5.25: Message delivery probability without the cure feature, using the Village mobility model.

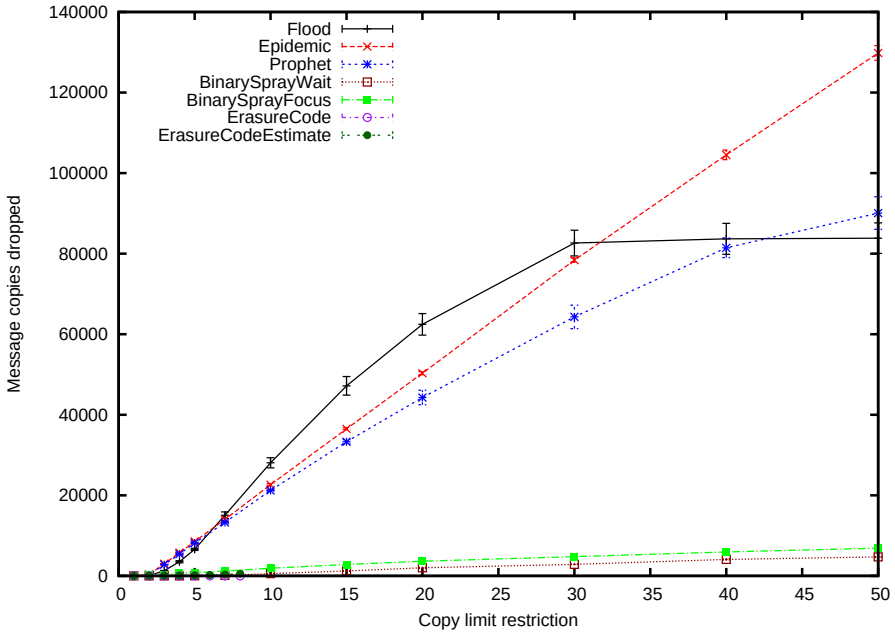


Figure 5.26: Number of message copies dropped without the cure feature, using the Village mobility model.

with Network Coding. Another restriction for the cure feature with Network Coding is that a message can not be cured until its entire generation has been decoded and delivered, further increasing the amount of delay between message delivery and the reclamation of buffer space.

5.7 The Impact of Hop Limits

The impact of imposing additional hop limit restrictions was the final architectural feature studied. The Epidemic, PRoPHET, Binary Spray and Wait, Binary Spray and Focus, and Estimation Based Erasure Coded algorithms were all tested. The Original Erasure Coded algorithm was not tested as it imposes its own two hop limit restriction, while Network Coding was not evaluated as it has no mechanism to restrict single messages to a predetermined number of hops. Figures 5.27 through 5.30 present the results of imposing the various hop limit restrictions for the Levy Walk mobility model. For the Epidemic, PRoPHET, Binary Spray and Wait, and Binary Spray and Focus algorithms, a hop limit greater than one was enough to achieve good performance, with increasing hop limit values doing little to improve it. For some of the algorithms, there was an additional improvement in delivery latency at three hops. These results imply that the majority of the messages can effectively be delivered by utilizing only two or three hops. Interestingly, the performance of Estimation Based Erasure Coding actually decreases with hop limit values greater than 2. This is related to the issue of non-reliable transfer of message ownership during the pure forwarding phase of this algorithm. A lower hop limit reduces the opportunities for a message to be dropped, therefore increasing its performance.

The hop limit restriction has been shown to be a very coarse-grained parameter to impact the behavior of these algorithms. The relative insignificance of hop limit values greater than three should be noted when analyzing any algorithm making claiming that its ability to forward across multiple hops is an important feature.

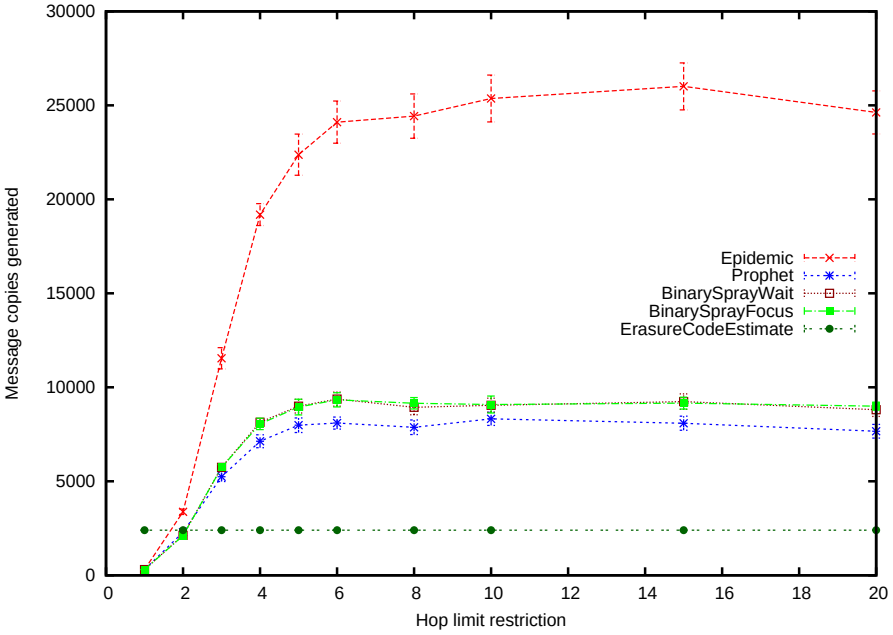


Figure 5.27: Impact of hop limit parameter on the number of messages copies generating, using the Levy Walk mobility model.

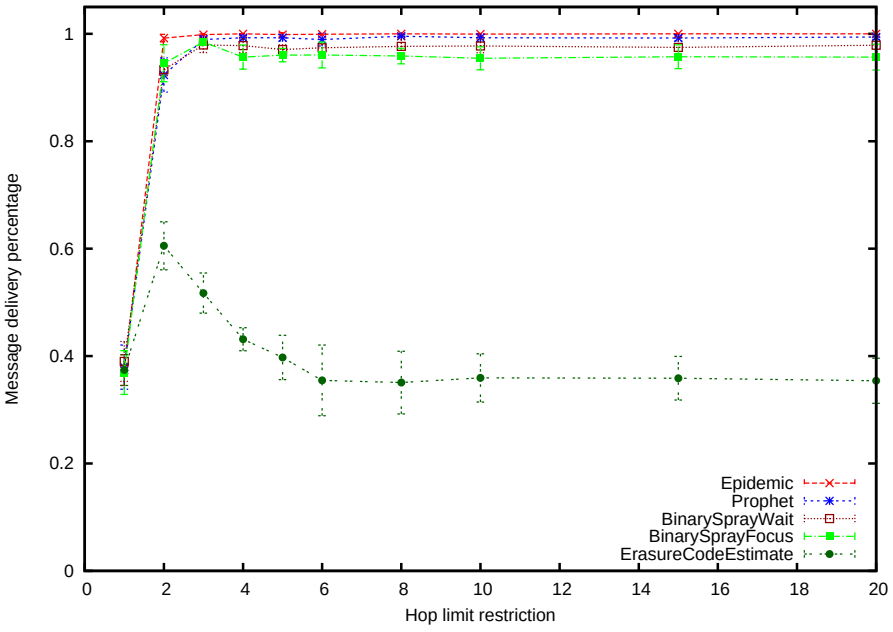


Figure 5.28: Impact of hop limit parameter on the number of delivered messages, using the Levy Walk mobility model.

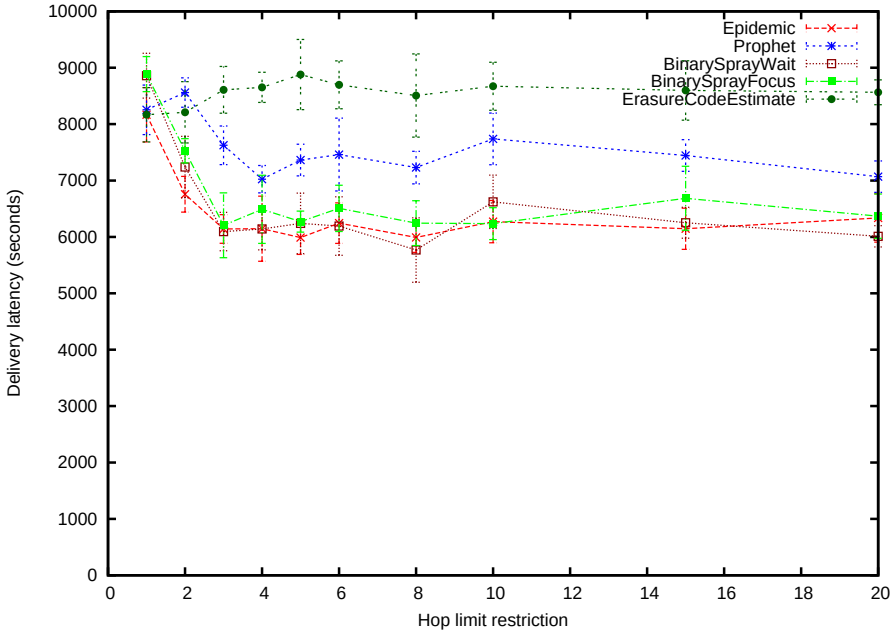


Figure 5.29: Impact of hop limit parameter on the delivery latency of messages, using the Levy Walk mobility model.

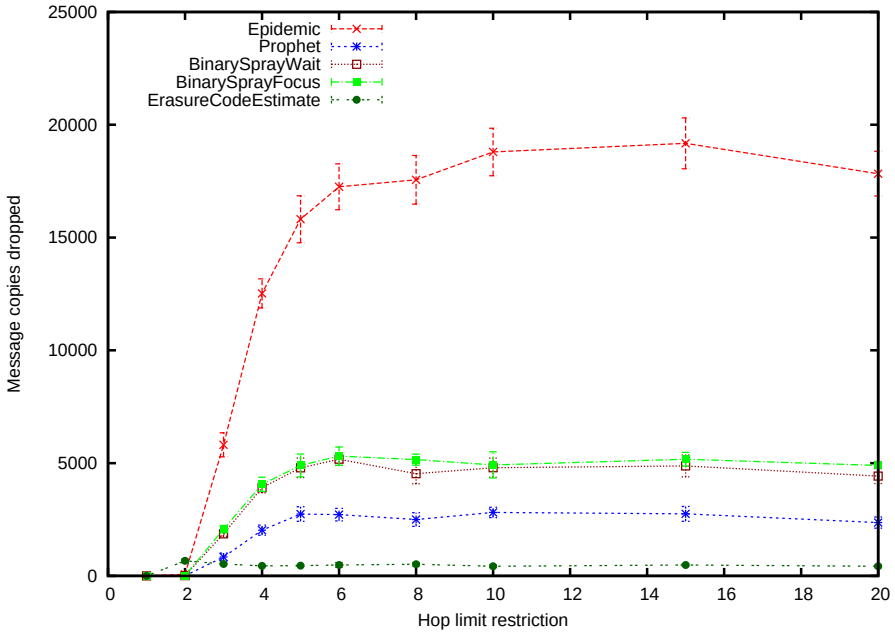


Figure 5.30: Impact of hop limit parameter on the number of messages copies dropped, using the Levy Walk mobility model.

As the mobility of hosts becomes more localized, it is likely that the number of hops required to achieve satisfactory delivery results will increase. It would be interesting to further study the relationship between mobility locality and the number of hops required for adequate delivery performance.

5.8 Summary

This section presented results evaluating the performance of Delay Tolerant Routing algorithms under a variety of conditions. By no means are these evaluations meant to be exhaustive, but instead they are meant to demonstrate that the simulation platform implemented can adequately be used to obtain interesting results. A number of surprising results were discovered, such as Flooding outperforming Epidemic, the explosion of messages with Network Coding, and the somewhat lackluster performance of the erasure coded algorithms. Both the routing architecture and the implemented simulation framework show considerable promise for the future study of DTN routing algorithms.

Chapter 6

Future Work

There are a number of areas throughout this thesis that could benefit from further investigation. Among them, the simulation implementation could benefit from a number of enhancements to have it better represent a real world system. There are also further areas of interest related to the characteristics and performance of the routing algorithms that were discussed and evaluated. Finally, the evolution and refinement of the routing architecture should also be explored.

Although considerable effort was made to make the routing algorithm implementations and the simulation framework as realistic as possible, a number of enhancements can still be made. Comprehensive regression tests for the different algorithms and mobility models should be put in place to better ensure new functionality can be implemented with confidence that old features are unaffected. Additionally, the size of the additional control information and data structures could be incorporated into the overall buffer space usage calculations, as the current approach only considers the size of the messages stored in the buffer. This would provide an additional dimension to the trade-off of one feature over another, as some features such as message curing require more storage of additional control information than others. Furthermore, the different entries in the control structures should be aged

and discarded after a period of time to reclaim space after the information is no longer useful. Furthermore, the current erasure coded and network coded algorithm implementations do not fully perform the coding operations as would be required in a real system. Although this is a reasonable approach for simulation, it does inhibit taking into account any additional time required during message exchange due to higher computational requirements of the coding procedures. This additional time may have a performance difference in highly mobile networks with very short encounter times.

One goal for this work would be to migrate the simulation implementation to a real world testbed. Because the simulation is implemented in C++, and it follows a simple interface with the layers above and below it in the network stack, it is conceivable that with some additional work these algorithms could be run in a real system with minimal implementation changes. One promising path forward would be to use the Click Modular Router [29] which provides a simple, modular interface to the Linux kernel's MAC layer, making it much simpler to integrate the simulation code into a real software stack. Leveraging the same code base for both simulation and real world testing would provide interesting insight into the fidelity of the simulation results and provide guidelines for improving simulation accuracy.

There are a number of areas related to the evaluation of the routing algorithms that could benefit from future work. Further research into DTN traffic patterns and their impact on algorithm performance would be interesting. The traffic generation for the evaluations presented in this thesis were random and somewhat arbitrary, which may not adequately represent any real world application patterns. Additionally, exploring the performance differences between a connection-oriented or connection-less implementation of `DtnProtocol` may also yield interesting insights. One approach or the other may be better suited for different environments, or it may be clear that one is generally better than the other in almost any situation.

Although the network coding algorithm is an interesting approach and shows promise, its current implementation needs some adjustment to be competitive with other algorithms in buffer limited situations. Further work needs to be taken to improve the exchange of summary information between hosts to prevent a proliferation of message copies that have little to no benefit. The primary issue with the current approach is that there is no exchange between hosts of the current available buffer space available nor any indication of the lowest priority message currently in a host's buffer. Armed with that knowledge, hosts could better evaluate whether a message in their buffer should be sent to a neighboring host, as they could determine if the other host's buffer is currently full and if so whether the message being sent has too low a priority for the other host to store it. Although this issue was primarily exposed with the network coding algorithm, it is a new feature that could provide benefit to any of the proposed protocols. Implementing this feature and understanding its challenging effects would be a worthwhile effort.

Finally, the simulation architecture itself should not be set in stone, but should continue to be refined and evolve as more fundamental features become apparent and new ideas are proposed. Maintaining a current summary of the architectural features available allows system designers to continue to have an up to date consideration of the tradeoffs of one feature over another as they work to design DTN systems for deployment.

Chapter 7

Conclusions

The routing architecture proposed by this thesis has been shown to provide a significant benefit toward the understanding and evaluation of Delay Tolerant Networking routing algorithms. Quite simply, routing algorithms are a complex combination of a number of features attempting to achieve the goal of delivering messages. This architecture provides a systematic way of breaking down the complexities of the combination of features, allowing them to be analyzed individually, instead of always as a pre-configured combination.

Throughout this work, I have come to the conclusion that it is extremely difficult to fully grasp an algorithm and its characteristics by only relying on published academic research. The presented results are too narrow, and there are too many implementation details that go unexplained and could be responsible for any significant difference in performance results. Additionally, the algorithms that are evaluated against one another often typically differ in multiple features, making it difficult to isolate which one(s) truly influenced the results. The simulation architecture that was designed and implemented such that it is much simpler to isolate the different features and their performance impacts, making it a very suitable platform for a more systematic evaluation of DTN algorithms.

The process of implementing algorithms provides a much deeper understanding of the routing algorithms and Delay Tolerant Networking in general that would be difficult to obtain otherwise. It was through the work of implementing the standard protocols that their underlying architecture first became apparent. Implementing a large variety of protocols is not always a practical endeavor for researchers. However, the work performed for this thesis to implement these protocols certainly yielded tangible results in the form of a much deeper understanding of Delay Tolerant Networking that would not have been gained otherwise, and is hopefully a benefit to the research community as well.

Bibliography

[1] The inet framework for omnet++. `http://inet.omnetpp.org/`, 2011.

[2] The ns-3 network simulator. `http://www.nsnam.org/`, 2011.

[3] The omnet++ network simulator. `http://www.omnetpp.org/`, 2011.

[4] Standard template library programmer's guide. `http://www.sgi.com/tech/stl/`, 2011.

[5] R. Ahlswede, N. Cai, S. Li, and R. Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 46(4):1204–1216, 2000.

[6] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla. Glomosim: A scalable network simulation environment. *UCLA Computer Science Department Technical Report*, 990027, 1999.

[7] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, et al. Advances in network simulation. *Computer*, 33(5):59–67, 2000.

[8] E. Brewer, M. Demmer, B. Du, M. Ho, M. Kam, S. Nedevschi, J. Pal, R. Patra, S. Surana, and K. Fall. The case for technology in developing regions. *Computer*, pages 25–38, 2005.

[9] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. Impact of human mobility on opportunistic forwarding algorithms. *IEEE Trans. on Mobile Computing*, pages 606–620, 2007.

[10] X. Chang. Network simulations with OPNET. In *Simulation Conference Proc., 1999 Winter*, volume 1, pages 307–314. IEEE, 1999.

[11] L. Chen, C. Yu, T. Sun, Y. Chen, and H. Chu. A hybrid routing approach for opportunistic networks. In *Proc. of the 2006 SIGCOMM workshop on Challenged networks*, page 220. ACM, 2006.

[12] P. Chou, Y. Wu, and K. Jain. Practical network coding. In *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, volume 41, pages 40–49. The University; 1998, 2003.

[13] P. Costa, M. Musolesi, C. Mascolo, and G. Picco. Adaptive content-based routing for delay-tolerant mobile ad hoc networks. Technical report, Technical report, UCL, 2006.

[14] D. De Couto, D. Aguayo, B. Chambers, and R. Morris. Performance of multihop wireless networks: Shortest path is not enough. *ACM SIGCOMM Computer Communication Review*, 33(1):83–88, 2003.

[15] A. Doria, M. Uden, and D. Pandey. Providing connectivity to the Saami nomadic community. In *Proc. of the 2nd Int'l. Conf. on Open Collaborative Design for Sustainable Development*, 2002.

[16] F. Ekman, A. Keränen, J. Karvo, and J. Ott. Working day movement model. In *Proceeding of the 1st ACM SIGMOBILE Workshop on Mobility Models*, pages 33–40. ACM, 2008.

[17] J. Eriksson, H. Balakrishnan, and S. Madden. Cabernet: vehicular content

delivery using WiFi. In *Proc. of the 14th ACM International Conference on Mobile Computing and Networking*, pages 199–210. ACM, 2008.

[18] G. Ewing, K. Pawlikowski, and D. McNickle. Akaroa-2: Exploiting network computing by distributing stochastic simulation. 1999.

[19] K. Fall. A delay-tolerant network architecture for challenged internets. *Proc. of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34, 2003.

[20] M. Grossglauser and D. Tse. Mobility increases the capacity of ad hoc wireless networks. *IEEE/ACM Transactions on Networking (ToN)*, 10(4):477–486, 2002.

[21] Z. Guo, G. Colombi, B. Wang, J. Cui, D. Maggiorini, and G. Rossi. Adaptive routing in underwater delay/disruption tolerant sensor networks. In *Proc. of the WONS*, pages 31–39, 2008.

[22] S. Hong, I. Rhee, S. Kim., K. Lee, and S. Chong. Routing performance analysis of human-driven delay tolerant networks using the truncated levy walk model. In *Proc. of MobilityModels*, pages 25–32, 2008.

[23] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot. Pocket switched networks and human mobility in conference environments. In *Proc. of the 2005 ACM SIGCOMM Workshop on Delay-tolerant Networking*, page 251. ACM, 2005.

[24] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. *ACM SIGCOMM Computer Communication Review*, 34(4):145–158, October 2004.

[25] A. Jardosh, E. Belding-Royer, K. Almeroth, and S. Suri. Towards realistic mobility models for mobile ad hoc networks. In *Proc. of MobiCom*, pages 217–229, 2003.

[26] D. B. Johnson, D. A. Maltz, and J. Broch. DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks. *Ad Hoc Networking*, 1:139–172, 2001.

[27] P. Juang, H. Oki, W. Want, M. Maronosi, L. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. *ACM SIGPLAN Notices*, 37(10):96–107, October 2002.

[28] A. Keränen, J. Ott, and T. Kärkkäinen. The ONE simulator for DTN protocol evaluation. In *Proc. of the 2nd International Conference on Simulation Tools and Techniques*, pages 1–10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[29] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[30] Y. Liao, K. Tan, Z. Zhang, and L. Gao. Estimation based erasure-coding routing in delay tolerant networks. In *Proc. of the 2006 international conference on Wireless communications and mobile computing*, pages 557–562. ACM, 2006.

[31] Y. Lin, B. Liang, and B. Li. Performance modeling of network coding in epidemic routing. In *Proc. of the 1st international MobiSys workshop on Mobile opportunistic networking*, page 74. ACM, 2007.

[32] A. Lindgren, A. Doria, and O. Schelen. Probabilistic routing in intermittently connected networks. In *Proc. of the 1st Int'l. Wkshp. on Service Assurance with Partial and Intermittent Resources*, pages 239–254, 2004.

[33] M. Mitzenmacher. Digital fountains: A survey and look forward. In *Information Theory Workshop, 2004. IEEE*, pages 271–276. IEEE, 2004.

[34] M. Musolesi, S. Hailes, and C. Mascolo. Adaptive routing for intermittently connected mobile ad hoc networks. In *Proc. of WOWMOM*, pages 183–189, 2005.

[35] M. Musolesi and C. Mascolo. A community based mobility model for ad hoc network research. In *Proc. of REALMAN*, pages 31–38. ACM New York, NY, USA, 2006.

[36] A. Pentland, R. Fletcher, and A. Hasson. DakNet: Rethinking connectivity in developing nations. *IEEE Computer*, 37(1):78–83, January 2004.

[37] C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. In *Proc. of the 2nd IEEE Wkshp. on Mobile Computer Systems and Applications*, pages 90–100, 1999.

[38] I. Rhee, M. Shin, S. Hong, K. Lee, and S. Chong. On the levy-walk nature of human mobility: Do humans walk like monkeys? Technical report, North Carolina State University, 2007.

[39] K. Scott and S. Burleigh. Bundle Protocol Specification. *IETF Draft, draft-irtf-dtnrgbundle-spec-01. txt, October*, 2003.

[40] K. Scott and S. Burleigh. Bundle protocol specification. *IETF Draft, draft-irtf-dtnrgbundle-spec-01.txt, October*, 2003.

[41] T. Small and Z. Haas. The shared wireless infostation model: a new ad hoc networking paradigm (or where there is a whale, there is a way). In *Proc. of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing*, pages 233–244. ACM New York, NY, USA, 2003.

[42] T. Spyropoulos, K. Psounis, and C. Raghavendra. Spray and wait: An efficient routing scheme for intermittently connected mobile networks. In *Proc. of WDTN*, pages 252–259, 2005.

[43] S. Sroka and H. Karl. Using Akaroa2 with OMNeT++. In *2nd International OMNeT++ Workshop, Berlin, Germany*, 2002.

[44] A. Vahdat and D. Becker. Epidemic routing for partially connected ad hoc networks. Technical Report CS-200006, Duke University, April 2000.

[45] A. Varga et al. The omnet++ discrete event simulation system. In *Proc. of ESM*, 2001.

[46] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke. Data collection, storage, and retrieval with an underwater sensor network. In *Proc. of the 3rd International Conference on Embedded Networked Sensor Systems*, page 165. ACM, 2005.

[47] B. Vellambi, R. Subramanian, F. Fekri, and M. Ammar. Reliable and efficient message delivery in delay tolerant networks using rateless codes. In *Proc. of MobiOpp*, pages 91–98, 2007.

[48] Y. Wang, S. Jain, M. Martonosi, and K. Fall. Erasure-coding based routing for opportunistic networks. In *Proc. of WDTN*, pages 229–236, 2005.

[49] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. *Peer-to-Peer Systems*, pages 328–337, 2002.

[50] J. Widmer and J. Le Boudec. Network coding for efficient communication in extreme networks. In *Proc. of WDTN*, pages 284–291, 2005.

[51] X. Zhang, J. Kurose, B. Levine, D. Towsley, and H. Zhang. Study of a bus-based disruption-tolerant network: mobility modeling and impact on routing. In *Proc. of MobiCom*, pages 195–206, 2007.

[52] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data

delivery in sparse mobile ad hoc networks. In *Proc. of the 5th ACM Int'l. Symp. on Mobile Ad Hoc Networking and Computing*, pages 187–198, 2004.