Routing in Partially Connected Wireless Ad Hoc Networks

Supervisor: Dr. Stephen Hailes

Group members: Vladimir Dyo
Melissa Ho
Kartik Muralidharan
Damir Pavusin
Jasper Weener

September 2003

Disclaimer: This report is submitted as part requirement for the MSc Degree in Computer Science at University College London. It is substantially the result of group own work except where explicitly indicated in the text.

The report may be freely copied and distributed provided the source is explicitly acknowledged.
Acknowledgements

We would like to acknowledge many people for their contributions to our project. First of all, our supervisor, Dr. Stephen Hailes, was an invaluable resource, both in terms of providing research materials and in terms of providing project checkpoints and sanity checks. Thanks also to Dr. Cecilia Mascolo for pointing us to several papers related to our topic.

In addition, we would like to thank Tracy Camp and the Toilers Group at the Colorado School of Mines for the use of their mobility models code.

Our project would not stand without the source code for the ns-2 extensions for epidemic routing, provided by Félix Hernández-Campos and Amin Vahdat.

Special thanks goes to Adam Greenhalgh for the use and abuse of his desktop for our simulations.

Melissa sends specific thanks to Samantha Cichero for the use of her computer every day while she was at work.
Routing in Partially Connected Wireless Ad Hoc Networks

Contents

ABSTRACT ............................................................................................................................. 4
CHAPTER 1: INTRODUCTION ............................................................................................... 5
CHAPTER 2: BACKGROUND .................................................................................................. 9
  2.1 EPIDEMIC ROUTING ................................................................................................. 9
  2.2 RELATED RESEARCH AND ALTERNATIVE PROTOCOLS ....................................... 12
  2.3 PROPOSED EXTENSIONS TO THE EPIDEMIC ROUTING PROTOCOL....................... 13
CHAPTER 3: ANALYSIS AND DESIGN ................................................................................. 14
  3.1 INITIAL ASSUMPTIONS .............................................................................................. 14
CHAPTER 4: PROJECT MANAGEMENT .............................................................................. 15
  4.1 INITIAL PLAN: ............................................................................................................. 15
  4.2 AN OVERVIEW OF THE XP PHILOSOPHY (SOURCE: EXTREMEPROGRAMMING.ORG) .......................................................... 16
  4.3 RISK MANAGEMENT ................................................................................................. 16
  4.4 PROJECT MILESTONES ............................................................................................. 18
  4.5 TEAM COMMUNICATION AND STATUS MONITORING ............................................. 20
  4.6 CRITICAL ASSESSMENT OF PROJECT MANAGEMENT TECHNIQUES ....................... 21
CHAPTER 5: SYSTEM EVALUATION .................................................................................. 23
  5.1 DESIGN ....................................................................................................................... 23
  5.2 NS IMPLEMENTATION ............................................................................................... 24
  5.3 SECURITY ISSUES OF THE EPIDEMIC ROUTING ................................................... 26
  5.4 MOBILITY MODELS ................................................................................................. 27
  5.5 VARIANCE REDUCTION ............................................................................................ 30
CHAPTER 6: DATA ANALYSIS ............................................................................................. 32
  6.1 BENCHMARKING ......................................................................................................... 32
  6.2 STEADY STATE RANDOM WAYPOINT ....................................................................... 40
CHAPTER 7: APPLICATION .................................................................................................. 44
  7.1 IMPLEMENTATION ..................................................................................................... 44
  7.2 GENERAL OVERVIEW OF SYSTEM FUNCTIONALITY ............................................. 44
  7.3 OVERVIEW OF SYSTEM COMPONENTS .................................................................... 46
  7.4 THE GRAPHICAL USER INTERFACES ......................................................................... 49
CHAPTER 8: CONCLUSIONS, EVALUATION AND FURTHER WORK ................................. 52
  8.1 CONCLUSION AND EVALUATION ............................................................................ 52
  8.2 FUTURE WORK ........................................................................................................... 53
APPENDIX A: SURVEY OF MOBILITY MODELS FOR EPIDEMIC AD HOC ROUTING .... 55
APPENDIX B: MOBILITY MODELS USER GUIDE ............................................................... 60
APPENDIX C: RUNNING THE SIMULATION AND GENERATING STATISTICS ............. 63
APPENDIX D: AREA OF APPLICATION ............................................................................ 65
APPENDIX E: SYSTEM MANUAL ....................................................................................... 67
APPENDIX F: DEVELOPER’S MANUAL ............................................................................ 69
  8.3 CLASSES ..................................................................................................................... 69
APPENDIX G: TESTING ..................................................................................................... 72
BIBLIOGRAPHY .................................................................................................................. 77
Abstract

Traditional mobile ad hoc routing protocols such as DSDV and AODV allow wireless nodes to communicate without pre-existing network infrastructure. Messages travel from neighbour to neighbour, following a connected path from source to destination. However, in the absence of such a path, messages cannot be delivered. Epidemic Routing, as described by Vahdat and Becker in “Epidemic Routing for Partially-Connected Ad Hoc Networks,” exploits the mobility of the nodes by allowing intervening nodes to buffer messages and “carry” them across breaks in the connected network. Thus, this protocol increases the potential robustness of ad hoc networks by providing a way to deliver messages when there is never a connected path between the source and the destination. This project seeks to continue the work begun in [Vahdat 2000] by validating their research in a number of different mobility scenarios, and extending Epidemic Routing with a number of modifications intended to improve performance. Namely, we examine the benefits of incorporating ACKs, broadcasting, counter-based timeouts, and the use of historical information to optimize use of buffer space. Through simulations using the ns-2, we show that our optimizations result in a more efficient use of buffer space, as well as an improvement in message delivery latency.
Chapter 1: Introduction

In this chapter we will make general introduction to the problem area (why is wireless networking interesting and how our research applies to the real world) and briefly explain what is ad hoc routing, what are the limitations and what is our proposed solution. Structure of the document is will conclude this chapter.

Wireless networks are more accessible to the wide range of users more than ever. Wireless network cards (802.11) and short-range network devices (Personal Area Network) can be found in many inexpensive personal digital assistants, laptops and desktop computers. Cheap and easy to install add on accessories (USB Bluetooth, USB 802.11, PCMCIA, CF cards, SD cards) are available for computers and PDA’s without built in wireless network.

Wireless network enables easy and simple way of connection between many devices. Wireless networks are easier to install since they do not require wire installations. Many companies are already using wireless networks in business environment. Most commercial wireless networks are running in infrastructure mode – computers with wireless networks are using wireless access points to connect to the rest of the network. The advantage of this approach is faster and easier network setup (less wires) and reliable and fast network with hierarchical structure. The disadvantage is that there is still some infrastructure necessary (wireless access points). Nominal range of wireless 802.11b station is 100 meters but in office conditions (indoors, walls, reflections, obstacles) it is almost always less than nominal range, often just 25 or 50 meters. In case of large office an area that means many access points (base stations) are necessary to effectively cover the entire area.

Mobile Ad Hoc Networks (MANET’s) are networks that allow mobile hosts to communicate without pre-existing network infrastructure. Some nodes in the network could be moving and all nodes are expected to assist in the routing of the packets. Two hosts can communicate with each other even if they are not in direct transmission range. In ad hoc network, any host can be used as intermediary router host. Wireless hosts will dynamically create temporary network without using any fixed network infrastructure. Due to the limited range of wireless network adapter, multiple “hops” might be needed for two hosts to communicate across network. Each node participates in ad hoc routing protocol that is used to discover multiple hop paths through the network.

Mobile Ad Hoc Networks can be used in situations where it is not possible or economically viable to set up fixed infrastructure network. For example, participants of the meeting wish to establish a temporary ad hoc network, students using laptops equipped with wireless network cards to participate in the lecture, battlefield situations where it is not possible to set up infrastructure, disaster situations where existing infrastructure is damaged or destroyed.

Mobile Ad Hoc networks can be used in corporate environments to extend coverage area and reduce number of necessary fixed base stations.
Routing in Partially Connected Wireless Ad Hoc Networks

Recent strong development of wireless 802.11 hotspots is another potential area of usage. Mobile Ad Hoc routing can be used to extend coverage area of the hotspot without costs related to installation of additional fixed infrastructure.

There are a number of different ad hoc routing protocols for wireless networks proposing to solve multi hop routing problem in wireless networks using different approaches and ideas. Some ad hoc routing protocols such as LAR [1], GPSR [2], and DREAM [3] assume existence of location information from GPS device and are using this information in routing.

Other recently proposed ad hoc routing protocols such as DSDV [4], TORA [5,6], DSR [7,8,9], and AODV [10] use some form of flooding. Although different optimizations are applied to make flooding more efficient this is significant research area.

However, common assumption behind traditional ad hoc networks is that there is always an existing connected path between a source and destination. Most existing ad-hoc routing techniques work well if there is connected (but unknown) path between source and destination host.

Although this assumption might be true in some situations, in many real life scenarios there are disconnected “islands” of mobile hosts that never have fully connected path between them. If there is a network partition between source and destination, most ad hoc routing techniques will not work. Nevertheless, through host movement some pairs of hosts (not necessarily sender and receiver) periodically and randomly come into communication range of one another.

Figure 1. Exchange of messages in epidemic routing through node mobility
This effect is used in Epidemic routing protocol [11] to enable routing even if there is never fully connected path between two hosts. In Epidemic, routing messages are distributed through connected parts of network. Messages are stored in buffer space for hosts that are not available at the moment but might be available in the future. Epidemic Routing then relies upon carriers coming into contact with another connected portion of the network through node mobility. At this point, through random pair wise exchanges the message spreads to an additional island of nodes. Through such transitive transmission of data, messages have a high probability of eventually reaching their destination. The overall goal of Epidemic Routing is to maximize message delivery rate and minimize message delivery latency, while also minimizing the aggregate system resources consumed in message delivery.

Epidemic routing algorithm is useful for some applications that do not rely on constant bit rate but might benefit from eventual timely delivery of the messages. For instance: asynchronous communication such as email communication, mobile sensor networks that are spread over large geographical areas but due to the available energy constraints single sensor unit have small transmission range, etc. Epidemic routing might be useful in commercial applications as well. A simple illustration where epidemic routing is used to extend mobile network even further can be applied to corporate environment or hot spot scenario mentioned above.

Our proposal aims to improve the performance of this algorithm by applying several techniques and heuristics. There are four major modifications to the original epidemic routing protocol:

1. Broadcast: leverage shared medium to improve performance by allowing neighbouring nodes to “overhear” pair-wise exchanges

2. ACKs: use receipts/acknowledgements of delivery to supplement message exchange session (anti-entropy session) with the exchange of a “message delivered” vector. This vector can be used to free the buffer space associated with messages that have been previously delivered.

3. Counter: number of counter based optimizations to reduce the overhead during broadcast, to optimize usage of available buffer space, etc.

4. History: our algorithm is based on a simple assumption that the near past can serve as an estimate of the nearest future. So, if we have contacted a certain node several times already, then we can assume there’s a high probability we will meet this node again soon. Accordingly, if we haven’t contacted a certain node in the nearest past, then we can assume there’s low probability we will meet this node in the future. We can use this algorithm to establish a smarter queue discipline and optimize the usage of buffer space.

We will evaluate the effects of each of these optimizations on the characteristic performance of the protocol, using the original protocol as a benchmark for performance and validate original research and our results by testing the protocol in different mobility scenarios [14].

We have decided to take epidemic routing as a base of our research because we believe routing in partially connected networks is largely Terra Incognita in
research terms. Epidemic routing was logical choice since it offers robust solution to the problem of routing in partially connected networks. However, we believe algorithm can be improved especially by applying garbage collection for more efficient usage of the resources and broadcast exchange of messages to exploit the natural properties of wireless media.

The rest of this paper is organized as follows: Section II – Background, in depth explanation of how epidemic routing protocol works and key characteristics. This chapter will include overview of the similar research projects and how they affected our work. Section III – Design, will discuss requirements for the protocol and reason about proposed improvements, explain theoretical background and list initial assumptions. This section will include detailed system structure. Section IV presents project management scheme and methods we used. Section V – Implementation and system evaluation specifies details about protocol implementation in NS, mobility models and variance reduction. Section VI – Testing, details tests we used and how the system was verified in NS and gives critical comparison of protocols for each mode. Section VII covers java implementation. Chapter VIII gives evaluation of the project as a whole and further work on protocol, analysis and application. Paper finishes with critical appraisal of the work and overall conclusion.
Chapter 2: Background

In this chapter we will introduce background information necessary for reader to understand the rest of the report. It will include detailed explanation of the existing epidemic routing algorithm and our proposed extensions. We will also mention relevant research in the field.

2.1 Epidemic routing

![Diagram of Epidemic Routing]

Existing ad hoc routing protocols are unable to transmit between a source and destination that does not have a connected path. Partially connected ad hoc networks on the other hand work with the assumption that there is never a connected path from source to destination or that a network partition exist at the time that messages originated.

Recent research in this field includes Epidemic Routing for Partially-Connected Ad Hoc Networks [11] where messages are distributed within connected nodes in a set of message exchange sessions. Through node mobility, carriers contact other connected portions of network and message is spread to additional island of connected nodes. Given random exchange of data among replicas, all replicas will see all updates in a bounded amount of time.

Epidemic routing protocol uses a variant of the theory of epidemic algorithms [12]. This theory states that given random exchange of data among replicas, all updates will be seen by all replicas in a bounded amount of time.

The goals of Epidemic Routing are to: efficiently distribute messages through partially connected ad hoc networks in a probabilistic fashion, minimize the amount of resources consumed in delivering any single message, and maximize the percentage of messages that are eventually delivered to their destination.[11]
Routing in Partially Connected Wireless Ad Hoc Networks

Some of the issues regarding the routing protocol in the epidemic routing are:

- **Routing under uncertainty** – since hosts have no knowledge of exact location of other hosts it is not possible to make a smart decision on weather to pass the message to the encountered host and if carrier host will actually be in position to ever deliver the message.

- **Resource management** – mobile hosts have limited buffer space. System must balance the conflicting goals of maximizing message delivery and minimizing resource consumption. There will most likely be more copies of the message in the system but it is not desirable or necessary for delivery to have a copy of the message on every host.

- **Performance** - storing and transmitting messages consumes energy as well as traditional performance metrics such as CPU cycles, memory, and network bandwidth, it is important to balance the consumption of all system resources in transmitting messages to their final destination and performance of the system (average latency, average resource consumption, average delivery rate).

- **Reliability** – given the nature of the epidemic routing it is not possible to give delivery guarantees. It is possible to introduce explicit acknowledgements of successful message delivery. Other hosts can free their buffers upon receive of acknowledgement.

- **Security** – a message will most probably traverse an arbitrary path of hosts before reaching destination. Different applications might require certain authenticity guarantees that can be provided using traditional cryptographic techniques. In addition, each message could include list of traversed hosts or include preferred trusted hosts (it will not be exchanged if receiving host is not in the list of trusted hosts)

The epidemic routing protocol works as follows [11]:

="Each host maintains a buffer consisting of messages that it has originated as well as messages that it is buffering on behalf of other hosts. For efficiency, a hash table indexes this list of messages, keyed by a unique identifier associated with each message. Each host stores a bit vector, called the *summary vector* that indicates which entries in their local hash tables are set.

When two hosts come into communication range of one another, the host with the smaller identifier initiates an *anti-entropy session* with the host with the larger identifier. To avoid redundant connections, each host maintains a cache of hosts that it has spoken with recently. Anti-entropy is not re-initiated with remote hosts that have been contacted within a configurable time period."
During anti-entropy, the two hosts exchange their summary vectors to determine which messages stored remotely have not been seen by the local host. In turn, each host then requests copies of messages that it has not yet seen. The receiving host maintains total autonomy in deciding whether it will accept a message.

For example, it may determine that it is unwilling to carry messages larger than a given size or destined for certain hosts.”

This process is repeated transitively when host encounters a new neighbour. Given sufficient buffer space and time, these anti-entropy sessions guarantee eventual message delivery through pair-wise message exchange [11].

A detailed step by step procedure would look like this:

1. Host A comes into contact with host B
2. Host A checks its counter to see if he has contacted host B in recent history, if so host A will not initiate anti-entropy session with host B. Host A will check trusted host policy (if existing) to determine weather it wants to communicate with host B.
3. Host A initiates anti-entropy session
4. Host with smaller ID (let us assume that is host A) transmits its summary vector first (a compact representation of all the messages being buffered) followed by the host with bigger ID (host B in our case).
5. Host B performs a logical AND operation between the negation of its summary vector and other hosts summary vector. It determines the difference set, messages that it needs.
6. Host B transmits a vector requesting these messages from host A
7. Host A sends requested messages to B
8. Host B sends requested messages to host A
9. Both host can independently decide which messages they want to accept (depending on buffer size, trusted host list, counter, etc.)
10. Both hosts update their counters (message hop counter, host history, etc). Message with hop count 1 will only be delivered to the destination host. Old messages will eventually drop out of queue.

Vahdat and Becker design of epidemic routing associates a unique message identifier, a hop count, and an optional ack request with each message. Message identifier is a concatenation of the host’s ID and a locally-generated message ID (16 bits each). Hosts have statically assigned ID’s.
Routing in Partially Connected Wireless Ad Hoc Networks

Hop count is similar to TTL field in IP packet. Message with hop count of one will only be delivered to the destination host and will eventually be dropped from the queue subject to the available buffer space. Messages with high priority can be assigned high hop count which will place them at the top of message buffer and ensure their faster delivery at the cost of extra resource consumption.

Message acknowledgements were not implemented in this version of epidemic routing protocol. Acknowledgements are simple return messages notifying the successful message delivery to the sender. Acknowledgements vectors can also be exchanged in anti-entropy session and serve as garbage collection tool to remove delivered messages from message buffers.

Each host sets buffer size independently. Buffer size limits the resources used in epidemic routing. Larger buffer sizes generally decrease message delivery latency at the higher cost (resources + network traffic). Very small buffer sizes result in old messages being flushed in favour of new messages before they can be delivered.

Current buffer queue strategy uses FIFO policy. Old messages will be flushed from the buffer once there are enough new messages in the system. FIFO strategy is reasonable in situations where buffer space is larger than number of messages in the system at any given time. Yet, in mobile networking area buffer space is always limited and FIFO strategy is suboptimal in regards to the QOS issues and per host fairness. Hosts with more traffic will occupy more space in buffer which will not be fair to other hosts. Fair queuing algorithms such as WFQ are proposed for future research.

2.2 Related research and alternative protocols
Gossip-Based Ad Hoc Routing [13] propose a gossiping-based approach, where each node forwards a message with some probability, to reduce the overhead of the routing protocols. In the networks we have considered, using gossiping probability between 0.6 and 0.8 suffices to ensure that almost every node gets the message in almost every execution. For large networks, this simple gossiping protocol uses up to 35% fewer messages than flooding, with improved performance.

Pure gossip protocol will not perform well in partially connected environments [2]. It performs well in connected ad hoc networks, where it is used to find yet undiscovered but existing paths in fully connected ad hoc networks [1]. In contrast, epidemic routing ensures eventual delivery of messages even if there is no connected path between the source and destination at any given point in time.

Toilers, A Mobile Ad Hoc Wireless Networking Research Group at Colorado School of Mines has done significant research in areas of mobility models[14], efficient broadcasting schemes in MANET’s and their comparison[15], Predictive Modelling of Network Wide Broadcasting Protocols for Mobile Ad Hoc Networks [16]

A Group Mobility Model for Ad Hoc Wireless Networks [17] discusses how various mobility patterns will affect the performance of different network protocols in different ways. By developing a flexible mobility framework that allows model different applications and network scenarios (e.g., individual and group; cellular and ad hoc, etc) it is possible to identify the impact of mobility on different scenarios.
Enabling Disconnected Communication [18] a document published at University of Rochester focuses on disconnected communication with a simple idea: rather than simply fail when immediate delivery is not possible, move the entire message (not a single packet) to another host as close to the destination as possible, where closer is defined to be a host which is likely to be in contact with the destination earlier than the source.

An efficient reliable broadcasting protocol for ad hoc networks [19] proposes using efficient reliable broadcasting to approach broadcast storm problems but differs from existing reliable broadcast solutions by adopting low cost unreliable broadcast as a basic operation.

An Experimental Study of Basic Communication Protocols in Ad-hoc Mobile Networks [20] follows the semi-compulsory approach according to which a small part of the mobile users, the support $\Sigma$, moves in a predetermined way and is used as an intermediate pool for receiving and delivering messages.

For complete listing of project reading links please refer to project website.

### 2.3 Proposed extensions to the epidemic routing protocol

We propose three changes to the existing epidemic routing algorithm that should reduce overhead and latency and minimize resources used.

**Garbage collection** - using receipts in opposite direction, spreading in epidemic manner. After successful message delivery to the receiving host there will still be a number of messages floating in the system. They will eventually die out (after a certain number of hops) but we hope to reduce their number by using receipts (or acknowledgements – ACK’S) sent from receiving host and spreading in the same epidemic manner. Since the ACK list is much smaller, we hope that exchanging list of ACK's for the already delivered messages will reduce traffic in anti-entropy session (message exchange session). Messages that have already been delivered will be pruned and will not be exchanged.

**Using broadcast instead of unicast in anti-entropy session.** Since 802.11 is already a broadcast media, everyone on the same channel overhears all communication (unicast, multicast or broadcast). Hosts within a single hop (within radio range) might benefit by overhearing this communication. However, it will increase receiver-on time and might introduce additional processing. We will try to find out optimal parameters and models for this strategy.

Use **utility models** to limit and direct spreading of epidemic messages to ensure faster delivery and limit resources usage. Number of history parameters are already stored (list of recently contacted hosts, list of messages, list of ACK's, timing information related to connection type and duration, number of hops for message or ACK...). We propose using stored information and possibly other information such as available memory, CPU power, and available battery power to differentiate hosts. The protocol will try to pass the message to the hosts most likely to deliver it and hosts with lowest cost. Hosts that have contacted destination host previously, or hosts in a destination host cluster (if destination host is in fully connected ad hoc group/cluster) will have a higher probability of contacting the destination host.
Chapter 3: Analysis and Design

3.1 Initial assumptions

- All hosts use 802.11b or 802.11g protocol (can establish ad hoc connection).
- All hosts must use common channel in order to establish ad hoc connection. We will propose single channel to be used. Single channel has theoretical bandwidth of 11MBps, which is shared between all hosts in ad hoc.
- All hosts are listening to all broadcasts at MAC layer. Hosts switch off its receiver between irrelevant broadcasts. We must research how much power will be used if receiver is on at all time (if there is traffic). How much more battery power will this mode of operation use compared to the unicast message mode?
- Power management in an ad-hoc environment works by all stations but one powering down between beacons, and a station that wants to transmit a frame waits for the time when all hosts are awake before announcing the frame. The receiver then stays awake to receive the frame. Broadcasts and multicasts work in a similar way.
- Table of ACK's is few orders of magnitude smaller than table of MSG's. Therefore, it is reasonable to store and exchange ACK's before actual message exchange and thus reduce message tables. It will speed up message exchange and reduce buffer space used for storing messages. (Messages that have corresponding ACK's can be pruned prior to message exchange.)
- Broadcasts are not acknowledged (unicasts are). Under heavy load, frame loss can be a problem that must be considered. Still, unicasts reach everybody anyway so why not let them make use of the information?
- 802.11 radio is by definition broadcast media. Current anti-entropy session uses unicast messages to exchange messages between two hosts in radio range. Unicast messages are sent over broadcast media which means that channel is used anyway (no other host can transmit at the same time even if unicast message is sent). All hosts in range receive the message at PHY/MAC level, but after examining header they decide to trash it because it is not intended for them. Receiver shuts down between beacons for a short period between beacons. We speculate that extra CPU cycles and battery power used to listen and process overheard communication is worth the effort and will reduce number of transmissions and help reduce resources used at hosts overhearing the communication (they will benefit from the overhearing the communication and prune delivered messages from buffers). We expect this to be extremely beneficial in environments where hosts are moving relatively fast. The sending host might not have enough time to send X number of separate unicast sessions but will have time for a single broadcast session.
- We assume that certain history data exists and we will use this data to create utility models per message for each host. We assume that this will be beneficial, even if it introduces extra complexity. We will test this in different environments and try to see if the benefits outweigh the extra complexity.
Chapter 4: Project management

With the set of goals and problems, we decided to focus on two aspects – producing a working application and coming up with a routing algorithm. Thus we approached the problem from a more manageable standpoint.

4.1 Initial Plan:

Start Simple and Work Up

On the routing algorithm side, we started by duplicating previous work, so we have a basis for comparison and a basis for development and in the case of the application we started with a prototype with minimal features.

Eventually, the two projects would meet up, and the Messaging Application will use the routing protocols we implement and develop. Each cycle/iteration will include new features such as security or even new approaches to the problem, such as further modifications to the algorithm we develop, or improvements to the network models and metrics we are using to benchmark the algorithm.

The eventual goal was to have a working algorithm, with statistics to demonstrate its performance, and a working, open-source, platform-independent, ad hoc messaging application, for which we can plug in any routing algorithm and which dynamically configures itself to the characteristics of the network it is in. Ideally it would also be able to accept functionality plug-ins, allowing file transfer, whiteboard conferencing, or other features. The application should be secure.

Team structure and Methodology

We started out with 2 project managers and a hierarchal structure. However as all members of the group were contributing to the project management we decided to drop the first approach and opt for a flat project structure. All members of the group were responsible for all aspects of the project management

As we had an odd number of people in the group (5), we decided to operate in two pairs, with a floater third, who can be shifted to either pair as resources are required. Pairs are established for the duration of each phase or cycle, as we see fit. As this is a research project and we all have strong opinions we have selected this approach to all for everybody’s’ ideas to be included rather than dictated. We considered that for a small group there was no need for designated project manager
Among the existing project management techniques we decided to follow the extreme programming (XP) methodology. Using an existing, already proven methodology generally lowers risk. We however did not revert to a pure XP approach. There were some aspects of XP methodology, which were not appropriate to our project such as the Customer, who no longer played an intricate role, and were modified as required during different stages of the project. We would incorporate additional stages to our cycle to suit our style of development and focus only on those aspects that were appropriate to our project.

4.2 An Overview of the XP Philosophy (Source: extremeprogramming.org)

We communicate with each other and our supervisor not only by talking face to face, but also through code, tests, planning, and user stories. The design is kept simple and clean, and our methodology kept simple with simple rules to follow. We get feedback by testing our software starting on day one. We deliver our system to the customers as early as possible and listen to what they say. By communicating with each other and the customers, keeping things simple, and relentlessly testing our code, we are able to courageously respond to changing requirements and technology.

Embracing Change

XP empowers your developers to confidently respond to changing customer requirements, even late in the life cycle. Agility permits rapid development with a constant eye on testing and reaching our end goals, as well as the ability to respond to changing requirements.

Teamwork

This methodology also emphasizes teamwork. Managers, customers, and developers are all part of a team dedicated to delivering quality software. XP implements a simple, yet effective way to enable groupware style development. Small groups (2-10) allow for rapid and effective communication and work best.

Testing

XP emphasizes not just testing, but testing well. Tests must be automated and provide a safety net for programmers and customers. Also, tests are created before the code is written, while it is written, and after.

4.3 Risk Management

For each risk we had to identify the probability of the risk and the impact it will have on a project and our response to this risk. The response could be prevention or we can mitigate the risk.
1. Technical failure

Designing routing protocol is a difficult task. Given the limited resources and time constraints it is possible that won’t be able to improve existing routing protocol(s). The probability was high and would have a large impact on our project. Also there was the risk of not succeeding in designing the messaging platform. There was low probability in this case because building an application is predictable task; we can do it incrementally and iteratively. Failure however would have large impact on our project.

*Response*: Address this problem first; set the time limit so we don’t waste time on it. If we have not been able to devise an improved routing protocol by the deadline we will use best existing routing protocol and focus on the application (lower risk of failure)

2. Two simultaneous goals

As we were working on two parts of the project simultaneously we were at the risk not being able to focus on either enough. This could result in the failure of either one of the parts or both in the worst case.

*Response*: In this we need to perform constant monitoring. The team should maintain sight of overall project goals in choosing priorities.

3. Missing the deadlines

As always one of the key risks in any project development is missing the deadlines. Often realistic goals are never kept which leads to this risk. Also the probability of this risk was relatively high and would have a great impact is not tackled.

*Response*: To encounter this risk we need to set the milestones, evaluate progress on regular basis and act before actually missing the deadline. Allocate more resources where required when needed. Maybe refine requirements or even split the task. If we miss the deadline then we will stop, look at why the deadline was missed considering the importance of that item to the project.

Possible resolutions include the following

- Drop the item
- Allocate more resources
- Extend deadlines
- Drop lower priority items
- Finalize the cycle as rapidly as possible even if it will not be 100% what we planned and move on
4. Re-inventing the wheel

With most research projects this is the most dreaded risk of all. In all possibility an idea which we come up may have already exist and maybe even published. There was also the chance of someone proposing the same solution while we were working on ours

Response: To keep this risk at bay we need to do effective Pre-reading and stay up to date with current research. Regularly consult with supervisor as he has a broader overview of the research area.

5. Lack of Project Manager

As we were a small group we decided group there was no need for designated project manager. Each member was highly motivated and has an interest in the project management as well as the success of the project as a whole. However there was the risk of the group falling apart without a manager to keep the members focussed and motivated.

Response: To counter the absence of a project manager there would be daily group meetings so that all members know every aspect of the project. We are a cohesive, small, integrated group of friends and must keep each other highly motivated and focussed.

4.4 Project Milestones

The following lists the set of Milestones the group initially planned to achieve in the given timeframe.

May

Application:

Basic Feature Set - By the end of this phase we should have a working prototype of an instant messaging application, working over a normal infrastructure-based network.

Pluggable Routing - Once we have the basic feature set, we will design the interface for plugging in routing modules. The first routing module will be the network-infrastructure based routing (RIP, etc). The second will be in conjunction with the Routing team, an implementation of Epidemic Routing

Routing:

Duplicate Epidemic Routing - Simultaneously with the previous phase, we will duplicate the simulation of epidemic routing, the existing solution we plan to improve upon later. We should be able to reproduce the results listed in the paper describing the algorithm.
Implement Epidemic Routing - Once we have duplicated the simulation environment, we will implement epidemic routing for live usage. At the end of this phase, we should be able to use epidemic routing to communicate between our laptops.

June/July

Application:

Pluggable Functionality - Focus on the application will shift to allowing plug-ins, including support for file transfer, security, auditing and tracking of messages, and other appropriate features. Fixing of bugs encountered from previous releases will continue, as well as development of further tests.

Feature Development - Once the plug-in interface has been designed, we will implement plugins for additional functionality. The user interface will be matured, and more instant messaging features will be added.

Routing:

Implement Routing Algorithm - This is the most risky phase of the project. We will design and implement a one or more routing protocols for partially connected networks, and then evaluate them in the same simulation environment used for epidemic routing. Should we find that we have been successful, we will continue working on this phase through July and August, further maturing our mobility and network topology modules. Otherwise, we will document our results and shift focus and resources to the application. Choosing a bail-out point early reduces risk to the project as a whole, since we realize that attempting to find a more efficient algorithm is very risky.

Joint:

Integrate Epidemic Routing Implementation - The two teams will then work together to integrate Epidemic Routing with the Application. At the end of this phase the application will be ‘released’ to the Routing team, for use throughout the rest of the project. Further releases may happen in future phases, but this will provide a baseline application for use in testing new routing algorithms.

Beta Release - At this point our application will be ready for initial public testing. We will work as a team to deliver a working beta, and establish a bug tracking system for our testers.

August
Application:

Plug-Ins, Testing, and Bug Fixing - Throughout August, we will be implementing more plug-ins, doing further testing, and fixing bugs reported by the beta testers. By the end of this phase we should be prepared for an application release, as well as beta releases of any new plug-ins.

Joint:

Application Release - If we have implemented a new routing algorithm successfully, we will also release the associated routing plug-in. This time will be focused on providing a clean release for public consumption, and establishing a way of maintaining the application and encouraging further development.

Evaluation and Conclusions - As we near the end of the project, we will gather together all of our documentation, and everything we have prepared thus far, and compose our final project report and presentation. This will include taking time to evaluate the course of the project as a whole, our results, and how effective XP and our team organization were. Furthermore, we will develop plans and suggestions for future research.

4.5 Team Communication and Status Monitoring

The initial plan was to meet and work together daily, with a short meeting at the beginning and a daily log produced at the end of the day. The daily logs was maintained by one of the group members with a different one each week. The logs contained decisions made during the day, minutes of meetings we had as a group as well as with our supervisor and other professors. At the end of the week a weekly status reports indicating progress and goals for coming week was produced by the secretary.

A mailing list was created including all the group members. This facilitated effective communication between group members regarding progress of individual tasks as well as effective communication between the group and the supervisor. Communication through the mailing list was also archived for future reference.

A group website was created (http://www.cs.ucl.ac.uk/Students/z15_5) and a CVS repository was setup for saving all relevant group work and research. CVS is the Concurrent Versions System (http://www.cvshome.org), a dominant open-source network-transparent version control system. CVS is useful for everyone from individual developers to large, distributed teams. Its client-server access method lets developers access the latest code from anywhere there's an Internet connection and its unreserved check-out model to version control avoids artificial conflicts common with the exclusive check-out model. CVS was used to track both the source code and website as they grew.
We also had a small ‘calendar’ section on the website where group schedules were put up and important dates marked. This helped the group focus on what needed to be done and most importantly when. If any group member had to take a leave of absence, this would be indicated in the calendar section and informed to other group members via the mailing list.

As it was very important not to stumble onto treaded grounds effective pre-reading was done. Initial suggestions from our supervisor as well as searches on the internet gave us a fair amount of reading. Links to all papers that group members needed to read was checked into CVS. The reading list was divided among group members who were responsible to write up a summary of the paper and indicate whether it was relevant for other members to read. This would prevent other group members from wasting time reading irrelevant material. Any new reading material found was checked in and was informed to other group members.

The website was not only a repository of group related work but also a source of information to anyone outside the group that wanted to learn more about ad-hoc routing. Our supervisor kept track of our progress through the website! The web space was divided into two regions. The first region contained general information about ad-hoc routing and related topics. This section could be accessed by anyone interested to know more about our work. The other section contained information used among group members such as the daily logs, status reports and other important documents. This section could not be accessed by all and was protected with user name and password.

Regular updates to the group repository resulted in an ocean of information about ad-hoc routing available through a single location. The website was also designed very well, allowing easy access to the different sections of the project.

4.6 Critical Assessment of Project Management Techniques

Based on the programming and management methodology outlined, the project was split into two parts-routing and application, and initially Vladimir and Kartik were to work on the routing protocol while Damir and Jasper on the application with Melissa as the floater who would shift between the two groups as needed. Life cycles were set to two weeks with deliverables at the end of each. To facilitate status monitoring a weekly rotating secretary was assigned who would maintain daily logs on group progress. The group decided to meet everyday at the university and work on their assigned task. Most decisions were taken on a consensus based on how strong the arguments were given. We continued to follow the flat hierarchical structure with no project managers although we were cautioned by our project management course supervisor Graham.

However as the project progressed, more resources shifted to the routing protocol as progress was slow and was more important in respect to the project. Most of the protocol documentation was done by Damir while Vladimir incorporated these changes into the ns code. Melissa and Kartik were responsible for testing and
analysis while Jasper continued to work alone on the application. This arrangement of responsibility was carried out for most of the project development.

Although, weekly meetings with the supervisor continued throughout the project development, daily group meetings did not as the group decided to concentrate on their individual tasks and decide whenever a group meeting was necessary. This was one of the changes made to the original plan of meeting everyday. To facilitate the status monitoring via the daily log, group members were to inform the current secretary of any progress or updates. Towards the last few weeks of the project however, daily logs and status reports were no longer maintained. Maintaining a life cycle of two weeks also proved to be difficult as work progressed slowly. Deadlines were more floating than constant.

Research updates on the other hand were regularly checked into the group website consolidating all information into one big reserve. The website allowed all members to keep track of the progress and was also very well designed. Although the initial idea of pair programming was never actually carried out, members of the group had to co-ordinate with one another in order to facilitate individual tasks. The group repository made the process of integration and testing easier as all related work was available in a single location. Regular testing was performed as initially decided which enabled critical faults to be detected and corrected.

The project management techniques set out during the course were not carried out in complete. Changes were made to accommodate the current working situation. Overall there was no particular management philosophy that reflected the groups’ style of working.
Chapter 5: System Evaluation

This section of the report seeks to explain our methods for evaluating our changes to the epidemic routing protocol, and our rationale for these methods. The original epidemic routing algorithm, as described in “Epidemic Routing for Partially-Connected Ad Hoc Networks,” by Vahdat and Becker, will hereafter be referred to as EPI. XEPI is the algorithm implementing eXtensions to EPI, including the use of broadcast and ACKs, and XEPI-H includes the further extensions to XEPI, incorporating the use of history information to optimize the usage of buffer space. XEPI-C makes use of counter-based timeouts to optimize broadcasts. Our primary goals are to realistically evaluate the routing algorithms, both on their own, and with respect to EPI.

To that effect, there are two stages to the data analysis. First, we will perform a comparative analysis between EPI, XEPI, XEPI-H and XEPI-C, highlighting differences in the characteristics of the algorithms. However, in order to fulfill our goal of realistic evaluation, we will also do a comparative analysis of the performance of each of the algorithms in different traffic models. Further discussion of the choices and characteristics of models chosen will appear in the following sections, as well as in

In order to evaluate these algorithms, we modified the original implementation of EPI, as provided by the authors of the paper. They implemented EPI using the Monarch extensions to the ns-2 packet-level simulator. [Vahdat 2001] Once the routing algorithm is implemented, the ns-2 simulator uses it to model communications between mobile nodes. Scenario files specify the movement of the nodes, and the communication patterns. We use various mobility models to generate scenario files, and continue to use the same communication pattern originally chosen by Vahdat and Becker.

The following sections further expand on the implementation of the system used to evaluate XEPI and XEPI-H.

5.1 DESIGN

We designed a new protocol which implements all of the optimisations proposed by our research. It’s mostly a modification of original epidemic routing protocol. In this chapter we address the protocol design issues as well as issues concerning the design of the ns-2 extension to implement the protocol.

We identified two major of issues with the protocol design. First, The implementation of the protocol has to be verifiable to be able to check that the implementation matches the specification. However, we postponed formal verification until later stages for the two reasons: the current goal of design is to evaluate the general ideas presented by our research and not produce a detailed and fully correct implementation of the protocol. Also, the protocol semantic is simple enough to check its correctness without applying to a formal specification. Second, the new
Routing in Partially Connected Wireless Ad Hoc Networks

protocol has improve the routing performance. In terms of our metrics it means higher delivery rates, lower latency and lower total overhead on the network.

When designing a protocol we made a number of assumptions:

1. XEPI will work on top of IMEP which will handle such low level functions as neighbour notification, packet aggregation and address resolution
2. XEPI will work over 802.11 wireless networks
3. XEPI will work inside a simulator only. For this reason we did not take any security consideration into the design process.

Protocol design technique

As the total number of changes to the protocol is quite significant it would be quite difficult to implement and test all protocol optimisations at once. For this reason, we used a component oriented design technique, where components implementing various functions are implemented separately and then integrated into one module. This approach significantly simplified the implementation and protocol verification.

The protocol was designed for the ns-2 network simulator. The major requirements for the design of the ns-2 extension are listed below in the order of priority:

1. The extension has to allow for performance analysis in order to find the optimal protocol parameters, trends and trade-offs.
2. The extension has to be extendable. New functionality such as new history analysis algorithm or new queue discipline should be added easily if needed.
3. The extension has to be modular. It should be possible to change protocol operating modes by specifying simulation parameters or compile options.
4. As the extension will be used for long simulations it has to be very memory efficient. A memory problem can become a bottleneck for running long simulations.

Design results

Full details about the implementation of the protocol is available from Developer’s manual (see Appendix) and detailed protocol specification (see Appendix). The current implementation can be verified by running the simulations and comparing them with the published data. As was specified in the requirements section the current implementation supports different modes of operation depending on the compile options and simulation parameters. New functionality added to the protocol generate new entries for the trace file which allows to measure the performance of the protocol.

5.2 NS Implementation

To evaluate the performance of our protocols, we chose to implement it in the ns-2 network simulator. Ns-2 is a packet level simulator which supports radio-propagation
Routing in Partially Connected Wireless Ad Hoc Networks

models with 802.11 MAC, ad hoc routing protocols and mobility scenarios which can represent node movement in an ad hoc scenarios. As a result, it is a very good environment of experimenting with ad hoc routing protocols in a highly mobile environment, and provides sufficient flexibly to evaluate our protocols in a variety of conditions. The original version of EPI was already implemented using Monarch extension for the ns-2 simulator and was later ported to ns-2 simulator. The authors provided the source for this later port to us, so we were able to implement our extensions to epidemic routing by modifying their implementation.

EPI was implemented on top of IMEP protocol layer, because it provides a number of useful low level functions such as keeping a list of current neighbours and notifying nodes about new neighbours and when nodes move out of range. It also reduces the total amount of wireless retransmissions by aggregating multiple packets intended for the same destination. [Vahdat 2000]

The wireless agent originally consisted of a message buffer, hash summary vector of all messages in the buffer and a code to carry out an anti-entropy session. The original implementation was flexible, allowing for different modes of protocol operation. For example, it was possible to specify whether to use hash vectors during anti-entropy sessions, whether to use hop count or maximum allowed copies to spread the limit of messages and so on. The detailed list of all protocol modes and compile options to enable them is provided in the appendix.

For XEPI, we also modified a piece of code responsible for message exchange by using broadcast instead of unicast when sending messages. In XEPI-C, the packet is allowed to be broadcast only if its packet counter is less than a certain threshold, as specified by a simulation parameter. The packet counter is incremented every time a node overhears a message from its neighbours. For the purposes of this evaluation, we have set the counter threshold to 2, so once two duplicate packets have been received, the packet will no longer be broadcasted until the counter is reset. The counter is reset every 5 seconds as described in the counter based optimisations section of the protocol chapter. The default values of counter threshold and a refresh time showed a good performance original simulations. However, the optimal value of those parameters has not been researched. The value of the counter refresh period is a simulation parameter, as described in the User Manual.

To implement acknowledgments we modified a packet format by adding a new field for a packet type. We also added a code to generate acknowledgments upon receiving a message and a code to perform garbage collection upon receiving a new acknowledgment. Acknowledgements are stored in a separate buffer and have a separate hash vector. Acknowledgements spread in the same epidemic manner as messages. When a node meets a new neighbour it starts broadcasting all acknowledgements. Their spread as well as the spread of the messages is limited by broadcast counters and a hop count. More detail about the algorithm is available in the analysis section. These modifications are incorporated into XEPI.
Finally, when implementing a history heuristics we defined a new class which collects and processes history data. Currently, this class keeps information about the N most recent contacts with other nodes and can calculate the least frequently contacted node from the list of destination nodes in the message buffer. The history class can be expanded very easily to collect and process information about messages, contact duration and other information.

As described in the protocol section the history information is used to perform a more intelligent queueing discipline. We have created a new function which processess the message queue according to 2 disciplines so far: FIFO and History. The function accepts a queueing discipline as a parameter and removes one message from the buffer. The function can also be easily extended to support tail drop, random drop and other disciplines. XEPI-H uses History, and XEPI and XEPI-C use FIFO.

In summary, the implementation of XEPI for ns has an additional code and data structures for supporting acks, broadcasts and history. Each of these functionalities can be selectively enabled or disabled using simulation parameters and compile options. We perform our analysis of each by using the configurations for XEPI, XEPI-C, and XEPI-H as described above. Using this implementation, we can now evaluate these protocols under a variety of ad hoc mobility models, as described in the next section.

5.3 Security issues of the epidemic routing

Epidemic routing presents new security threats to an already wide range of security issues related to mobile ad hoc networks. We identify some of those threats and propose possible solutions by suggesting protocol modifications and using cryptographic techniques.

In epidemic routing every message is effectively spread over the entire network which can be partially-connected. When sending a message a node broadcasts the message to all of its neighbours while the message is in the buffer. The neighbours, in turn, spread the message in the same way. As a result the destination node often receives multiple copies of the same messages from several nodes. This makes the epidemic protocol very robust and not vulnerable to malicious or non-cooperating nodes which do not forward the message.

As each epidemic message is spread over entire network it’s possible for a malicious node to monitor the content of all messages between any two nodes in the network.
The message content can be protected by using encryption at the transport or a higher level protocol.

A malicious node can generate spoof acknowledgement which will purge the message from the network before it reaches the recipient. It can even generate spoofed acknowledgements with random message IDs before the actual message is sent to the network. Such an attack can significantly reduce the network message delivery rate. Here we describe a method proposed by Dr. Stephen Hailes during one of our group meetings.

In this solution a message header should contain an hash value of the confirmation ID which is contained inside the encrypted message itself. Upon the reception of the message the legitimate recipient decrypts the message, extracts the confirmation ID and sends an ack containing a message ID together with this confirmation ID. The node receiving an ack will use it for garbage collections only if the calculated hash value of the confirmation ID matches the hash value from the message header. If an ack is spoofed the calculated hash value of confirmation ID will not match the hash value in the header of the message.

Another problem with the epidemic routing is that it allows to easily to track down location of other nodes. Every node can see the traffic pattern between any two nodes in the network and the source and destination of each message. Malicious nodes can monitor freely the movement pattern of communicating nodes as all messages contain an entire message path. One of the solutions, suggested by our supervisor relies on the existence of some trusted nodes in the network serving as anonymous gateways. All nodes in the network will have to have a SA with anonymous gateways. If a node needs to send an acknowledgement or a message it send it to a gateway through an encrypted tunnel. The gateway will specify itself as a source and send the message to its final destination. There can be multiple anonymous gateways in the network and a node can send a copy of each message to several gateways at once. The same technique can be applied to hide the location of the destination node when sending an acknowledgement.

The problem of security in epidemic routing needs further exploration. There are many other potential threats which we did not research into due to time constraints.

5.4 Mobility Models

One of the main difficulties in defining realistic mobility models is that there is no realistic data available on ad hoc network structure and mobility. One can only hope for a reasonable approximation, based on current research and reasonable assumptions.

Current research uses fairly simplified mobility models, varying node speed and direction according to various algorithms. In fact, most of the primary related
research uses variants of the Random Waypoint model, in which each node picks a random spot, moves there with some fixed random speed, and pauses for a random amount of time [Haas] [Chat] [Vahdat]. The speed and the pause times are chosen in a uniform distribution over some pre-specified range. For example, in [Vahdat 2001], the simulations are run using the this mobility model, with speeds uniformly distributed between 0-10 m/s, and no pause time. Since we are using their algorithm as a benchmark for comparison, we use the same mobility model with the same parameters to evaluate XEPI.

See "A Survey of Mobility Models for Ad Hoc Network Research," by Camp, Boleng, and Davies [Camp 2002] for a detailed evaluation of several mobility models, including some group models. Our goals in selecting mobility models are to better validate the results generated for the original paper, as well as validating our own results.

In that vein, using Random Waypoint leaves much room for improvement. Although it is simple, there are several problems and inaccuracies inherent to using this model, which are not addressed by the writers of the paper. Namely, after a time, mobile nodes in this model reach a steady state, in which the nodes are more concentrated in the center of the specified region, rather than being uniformly distributed. Thus, while the nodes are settling into this state, the first n seconds of the simulation are invalid, or at the very least are not representative of how nodes will interact in longer simulations.

William Navidi and Tracy Camp's paper, "Stationary Distributions for the Random Waypoint Mobility Model," [Navidi 2002] asserts that a node traveling at an average of 10 m/s (+/- 10m/s) will not achieve stability for approximately 100 seconds. Their recommendation is to either throw away the first 1000 seconds of the simulation data, or to start the simulation with positions from the steady state, rather than the traditional random distribution. In the existing research, most messages are delivered within the first 1000 s of the simulation [Vahdat 2000], so the results may be considered to be questionable. In an effort to validate the original epidemic routing research in the light of this information, we evaluate the performance of the variants of our protocols using Camp and Navidi's "Steady-State Random Waypoint" mobility model, in addition to reproducing their results from the Random Waypoint mobility model.

In order to evaluate the merits of using history to improve the performance of ad hoc epidemic routing, we also propose a new mobility model, Multistage Random Waypoint. The previous models have a single state memory – since the movement is random, there is no history inherent to the models. To create a history, while maintaining a random element, we propose a model in which individual nodes have different characteristic patterns of travel:

- Nodes may stay within a certain locality, or range everywhere
- Nodes will have different typical traveling speeds
- Nodes may change pattern of travel over time

Thus, the model consists of several groups of nodes, with each group of nodes adopting a particular movement pattern. Patterns in how each group interacts serve to create a history within the model. Nodes that stay within the same locality will be
Routing in Partially Connected Wireless Ad Hoc Networks

seen more often, and nodes that roam everywhere, will tend to see more nodes overall. From a macroscopic level, the analogy explaining these chosen characteristics is people in various cities. Most people will stay within the locality of where they, but there are also people that tend to travel more, whether on business, or just as a choice of lifestyle. Within larger cities, people tend to move around faster, and sometimes people just pick up and move to a new city entirely. Within a wireless ad hoc context, mobile users tend to move around in the context of their home or office, and a subset of users will be “en route” at any given point in time. Epidemic Routing is specifically intended to exploit this feature – two semi-stationary nodes with no particular available path between them, could use an intervening mobile node to “carry” a message between them. Thus we are exploiting mobility as a means of extending the range of communication between various nodes.

For the purposes of studying epidemic routing, we will simplify this proposed model, and adopt the scenario most commonly used to illustrate epidemic routing. There will be three groups of nodes. Movement of the nodes within a particular group is governed by the Steady State Random Waypoint model. The nodes in the first two groups will be localized to two separate areas within the movement field, and the nodes in the third group, the “roamers,” will range over the entire field. If the first two groups are positioned such that their member nodes do not come within each other’s ranges, then the “roamers” will have to carry messages from one group to another. In this scenario, epidemic routing permits communication between the two groups where, due to frequent movement, there may not be a fully connected path. So, with the three groups having different parameters, we will be able to simulate a “history” in which the roamers will reappear periodically for each of the nodes in the localized groups.

Simulations were done using the four mobility model configurations: randway, ss, multi, and small. All nodes have an average speed of 10 m/s, with a delta of 10 m/s. This means that the speed is uniformly distributed between 0 and 20 m/s. The simulation area is 1500 x 300 m in dimension, as used in [Vahdat 2000], and the pause time is 0 s. A total of 1980 1KB messages are sent, one per second, for 1980 seconds.

For randway and ss, the simulation uses 50 nodes, also as used in [Vahdat], and 45 of the nodes send messages to 44 other nodes in the system, for a total of 1980 messages. Randway uses the Random Waypoint mobility model, and ss uses Steady State Random Waypoint. Both of these are generated using slightly modified versions of code provided by the Toilers Group at the Colorado School of Mines, as described in [Camp 2002] and [Navidi 2002].

Multi and small both use Multistage Random Waypoint, and are generated using our implementation of this model, incorporating code from the Steady State Random Waypoint implementation. Each simulation has 60 nodes, with 20 nodes per group. During the simulation, each of the nodes in the localized groups sends messages to each of the nodes in its own group, each of the nodes in the other localized group, and 5 messages to the roamers, for a total of 1980 messages.

Multi localizes one group to the left half of the simulation area, and the other group to the right half (See Figure ). Thus if any nodes are near to the center of the simulation area, then they will be able to communicate directly with members of the
other group, but in general messages will need to be carried from one group to another by the roamers. Note that in the steady-state distribution of Random Waypoint, nodes tend to concentrate within the center of the region [Navidi 2002], so most messages will be carried by the roamers or not at all.

**Small** is an attempt to study the behavior of epidemic routing in a denser network, where communication within the group does not necessarily require epidemic routing, but communication between groups may. The simulation field is 300 x 600 m, instead of 300 x 1500 m. Each of the localized groups occupies a square area, 200 x 200 m, in the center of its half of its side of the field. Thus, the minimum distance between nodes in the two groups is 100m, just outside of the 50 m range. This simulation is being done in part due to the inability to duplicate results when the range is 100 m or 250 m. Since the larger range is effectively higher density, reducing the scale amounts to a similar effect.

Details of the programs and parameters used to generate the mobility scenarios can be found in **Appendix B: Mobility Models User Guide**.

5.5 **Variance Reduction**

Another aspect of validating results is to generate the statistics over several simulations, and to use average values and variance as an indicator of the precision of the values observed. In order to do this properly, we created a series of automated scripts that would generate scenario files, run the simulation with the specified parameters, calculate the relevant statistics.

In terms of iterative and incremental development, this was done in several stages, as can be observed by the variety of scripts in use. Each script is modular and performs a separate function. As each script was completed, more scripts were written to automate the calling of the more specific scripts. The final product is a master script that generates both the trace files and the statistics, as various options are specified. Details about this script (runall.pl) can be found in **Appendix C: Running the Simulation and Generating Statistics**.

One of the key aspects of running the simulations was to appropriately generate different mobility scenarios for each simulation. Primary goals in designing the simulation runs were to make sure to be able to reproduce results later, and to make sure that the random number streams were not correlated in such a way as to introduce a relationship that is not really there. To that effect, we modified the mobility model generators, adding an additional argument to allow the scripts to specify the seed to be used in generating the random motion for the mobility model. By doing so, we are able to reproduce any given mobility scenario file, by specifying the known starting seed. In this way, the same set of scenarios was used for each configuration. In addition, we used a separate stream to generate the seeds used to generate the scenarios themselves, so we could easily extend the number of duplicate runs, without having first to define a set of known seeds. By using a pseudo-random number generator, we are again able to reproduce that set of numbers at will.

Once we have a set of simulation results, we use them to generate a set of statistics for each simulation. The variance calculation script takes the statistics from a set of simulations (using the same configuration and mobility model, but different specific
mobility scenarios generated with the specific set of seeds) and calculates the variance and standard deviation over each of those values.

In the interests of time (and because ns likes to crash for simulations longer than 4500 s), all simulations are run for 4000 seconds. As a result, some statistics will vary from those presented in [Vahdat]. Specifically, average latencies for which their max was over 4000 will be lower, since those times will not be included in the calculations. In addition, the delivery rate may be lower – as they list eventual delivery rate over 200,000 seconds (their max latency for 10 m range is 198,107 s), rather than the delivery rate at 4000 s. In terms of validating their results, then, we are comparing to see that we achieve the same trends.
Chapter 6: Data Analysis

6.1 Benchmarking

Our primary goal in data analysis is to compare the performance of XEPI and its variance to the performance of EPI. To that end, we have examined the characteristics of each of the protocols in a number of different scenarios, using the same metrics to measure the performance as detailed in [Vahdat]. We perform the initial comparison using results generated with the Random Waypoint model.

First, we examine the performance of EPI and XEPI at different radio transmission ranges. The following statistics are based on simulations using a buffer size of 2000 (infinite buffer space) and a max hop count of 4 hops. The following graph is the cumulative distribution function (CDF) of message delivery latency for transmission ranges of 10, 25, and 50 m. The x-axis shows the latency in seconds on a logarithmic scale. Given a longer simulation time, the line for 25 m (in the center) would also approach 100%.

![Graph showing message delivery latency for different radio ranges.](image)

**Figure 1: Delivery Rates of XEPI at Different Radio Ranges**

This next table shows the actual statistics derived from the simulations. Both EPI and XEPI achieve 100% delivery when the range is 50 m, a significant improvement over routing protocols that require a connected path from route to destination. Given the node density and coverage, traditional ad hoc routing protocols would be unable to find a connected path from route to destination, and estimated statistics for delivery rate is less than 1%. However, as you can see from Table 1 and Figure 2, there is no significant difference between the performance of EPI and XEPI for different transmission ranges, when using an infinite buffer size. Thus the addition of ACK
Routing in Partially Connected Wireless Ad Hoc Networks

and broadcast will not be useful if the limiting factor is insufficient range to cover the area in question. This makes sense, since in low coverage areas, the probability of the presence of two neighbors, one to initiate message exchange, and one to “overhear” the broadcast, is very low. In addition, any benefit garnered by the use of ACKs first requires that the messages are delivered. Once that happens, the ACKs will have the same delivery problems as the messages themselves, and will not provide much benefit since they will most likely not reach many other nodes.

Note also that the figures in the following table are different from those published in [Vahdat]. Although we are using the same mobility model and parameters, there are two key differences. Firstly, we are stopping the simulation after 4000 s, as stated before. This will lower the delivery rate, as any messages that would have been delivered after 4000 s (2000 s after the last message was sent) will not be delivered.

Table 1: Characteristics of EPI and XEPI as a Function of Transmission Range

<table>
<thead>
<tr>
<th></th>
<th>EPI</th>
<th></th>
<th>XEPI</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 m</td>
<td>1.2 % ± 0.2</td>
<td>86.6 % ± 6.0</td>
<td>1.3 % ± 0.2</td>
<td>86.5 % ± 6.6</td>
</tr>
<tr>
<td>25 m</td>
<td>100.0 % ± 0.0</td>
<td></td>
<td>100.0 % ± 0.0</td>
<td></td>
</tr>
<tr>
<td>50 m</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delivery Rate</td>
<td>1714.2 ± 220.6</td>
<td>1250.9 ± 4.3</td>
<td>1737.6 ± 294.0</td>
<td>1258.5 ± 37.4</td>
</tr>
<tr>
<td>Latency (s)</td>
<td>Avg 3327.09</td>
<td>3048.14</td>
<td>643.51</td>
<td>3636.2</td>
</tr>
<tr>
<td></td>
<td>3327.05</td>
<td>639.91</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hops</td>
<td>Avg 1.2 ± 0.2</td>
<td>2.8 ± 0.0</td>
<td>1.3 ± 0.2</td>
<td>2.9 ± 0.0</td>
</tr>
<tr>
<td></td>
<td>Max 5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Next we compared the performance at different maximum hop counts, using a transmission range of 50m, and an infinite buffer capacity. We choose a 50m because it achieves 100% delivery in a reasonably short amount of time, while also...
taking advantage of the ability to use partially-connected routing. Figure 3 shows the performance of XEPI at maximum hop counts of 1, 2, 3, 4, and 8 hops. In this particular mobility scenario, there is a slight improvement in performance if the hop count is increased from 2 to 3 hops, but there is no significant difference for maximum hop counts of 3, 4, or 8 hops. This correlates with the data from Table 1, in which the maximum number of hops was consistently 3 or 5 hops, with an average of 3.0 hops from source to destination. Therefore, since most messages only require 3 hops to be delivered, increasing the hop count further provides little benefit.

Overall, the performance characteristics of XEPI look very much like those of EPI. Given an infinite buffer size, ACKs will not have any impact on buffer usage itself, but may actually increase delivery latency due to increase in traffic. However, the small size of ACKs means that the impact on traffic should be negligible compared to the message traffic. On the other hand, the use of broadcast instead of unicast for message exchange may actually decrease message delivery latency, as nodes will potentially receive messages earlier than they would have normally. In addition, nodes that do not have time for message exchange, but are in range long enough to receive the broadcast, may actually receive messages it wouldn’t have gotten before. These potential gains are illustrated in Figure 4. There is a clear trend in which more messages arrive within 300 seconds when delivered using XEPI than when delivered using EPI. At a maximum hop count of one, the difference is present, but not significant, which indicates that messages are delivered in roughly the same amount of time from neighbor to neighbor. There is also not a significant difference when the max hop count is four. This indicates that the performance improvement gained by allowing the message to traverse more nodes (and thus providing more partially-connected coverage) far outweighs the performance gain from using XEPI. This difference is most pronounced when the maximum hop count is 2 hops, so when the
coverage is not quite sufficient for messages to be delivered in two hops, destination nodes bordering the 3 hop radius range receive the message in two hops, entirely due to the use of broadcast instead of unicast. This may happen because the destination node only comes in contact with the carrier node long enough to receive the broadcast of that message, as sent to another node, and not long enough for the anti-entropy session normally required before exchanging messages.

Table 2 shows the average delivery latency for both EPI and XEPI at different maximum hop counts. To be precise, this value is the average of the average values obtained from each individual simulation. The standard deviation, then is the deviation from simulation to simulation of the average message delivery latency. Since individual latency times vary widely, as exhibited in the cumulative distribution function of the latencies, the variance over the measured statistic is quite high. However, from simulation to simulation, the average latency is relatively stable, showing that the average of the average latencies can be a relatively effective indicator of performance, if not a guarantee of actual realistic performance. From the data here, we can gather that after 2000 seconds of simulation, there is no significant difference between EPI and XEPI. This correlates with the graph above, in which the lines for EPI and XEPI at 2 hops and 4 hops converge before 600 s, which is approximately when all messages have been delivered.

Table 2: Average Delivery Latency for EPI and XEPI at Different Max Hop Counts

<table>
<thead>
<tr>
<th>Max Hop Count</th>
<th>EPI</th>
<th>XEPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 hops</td>
<td>746.6 ± 52.6 s</td>
<td>750.0 ± 57.9 s</td>
</tr>
<tr>
<td>2 hops</td>
<td>202.0 ± 20.5 s</td>
<td>192.7 ± 19.9 s</td>
</tr>
<tr>
<td>3 hops</td>
<td>145.7 ± 13.6 s</td>
<td>143.6 ± 13.0 s</td>
</tr>
<tr>
<td>4 hops</td>
<td>134.4 ± 12.5 s</td>
<td>134.9 ± 12.0 s</td>
</tr>
<tr>
<td>8 hops</td>
<td>136.7 ± 13.5 s</td>
<td>139.5 ± 14.8 s</td>
</tr>
</tbody>
</table>
Thus far, the impact of our extensions to EPI have been fairly minimal. All differences in the statistics shown at this point have been well within an accepted margin of error for each protocol. Thus we have no conclusive evidence that extending EPI with ACKs and broadcasting provide any significant gain, especially given a primary goal of maximizing eventual delivery rate. However, all of the experiments thus far have involved using an infinite buffer space, a condition that makes ACKs unnecessary. The purpose of ACKs is to provide a garbage-collection mechanism for freeing up buffer space, so nodes can know with assurance that a given message no longer needs to be in its message queue. If the buffer size is virtually unlimited – if each node can carry all of the messages within a system, then this garbage collection is unnecessary and only provides more overhead. However, one can reason that this type of situation is unrealistic – any networked system will not have a fixed amount of messages to send, and thus ACKs become a necessary addition for recycling the buffer for newly originated messages, as messages constantly roll through the buffer queue. So, ideally, we would run the simulations for some amount of time, using a message model in which messages are continually sent throughout the lifetime of the simulation. In this case, we would expect some number of messages to be undelivered at any given time, but we could evaluate the steady-state performance within that particular model.

Unfortunately, limits of time and resources dictate that we evaluate our protocol under a simpler model. For the purposes of this section, we continue to duplicate the evaluation performed for the original implementation of EPI. Thus, messages are only sent for the first 1980 seconds, after which we allow the simulated network to “settle.” This gives the last few messages some time to propagate through the system. Thus, in order to make the ACK component of XEPI useful, we need to set the buffer sizes such that the recycling of the message becomes necessary, and examine patterns of behavior. The following experiments examine the performance of XEPI in a variety of buffer sizes, from 10 messages, up to 2000 messages. We expect XEPI to have little effect when the buffer size is 2000 messages, as evidenced by the previous investigations. However, for smaller buffer sizes, queue management issues come into effect, and we expect there to be some performance gain as the buffer size decreases.

Figure 5 and Figure 6 show the cumulative distribution functions of the message delivery latency for XEPI and EPI, for buffer sizes of 10, 20, 50, 100, 200, 500, 1000, and 2000 (unlimited). As expected, the performance characteristics are very similar in shape – with an initial sharp rise over the first 300 seconds, and then a flattening out for messages that are undeliverable, even with epidemic routing. Note that at this range, 50 m, traditional ad hoc routing protocols, such as DSR and DSDV, are unable to deliver most messages due to partial connectivity, whereas with EPI, we are delivering at least 30% of the messages at worst case. As shown, infinite buffer space results in 100% delivery, with only a slight loss in performance for buffer sizes down to 200 messages. This is true for both EPI and XEPI, although XEPI exhibits a slightly smaller performance loss – the curves for 200, 500, 1000, and 2000 messages almost overlap at this granularity.
Another interesting feature to note is that the percentage of messages delivered actually rises faster, and eventual message delivery rate is significantly higher, when using XEPI instead of EPI. The use of ACKs provides a clear performance gain when buffer size is limited, with larger gains with smaller buffer sizes. This indicates that individual nodes have more space for the more relevant messages – the undelivered ones, and are able to deliver them in a more timely fashion. By discarding delivered and acknowledged messages, each node buffers more undelivered messages and has the potential to carry them epidemically to other nodes.

Table 3 shows the actual delivery rates, as well as the buffer utilization characteristics for the different buffer sizes, when using a maximum hop count of 4 and a transmission range of 50m. The first column indicates the size of the buffer used for the simulations. Then there are two sections, one for EPI statistics, and another for XEPI. The first column in each section is the average eventual delivery rate after 4000 s. XEPI consistently shows better delivery rates than EPI, with improvements ranging from 0.6% to 7.1%. The margin of error is quite small, indicating that these statistical improvements will remain, even if more simulations are run. Currently, these averages are drawn from 3 simulation runs. Averages drawn from similar simulations, but with up to 30 runs, show similar variances. When the buffer size is set to 10, the delivery rate increases by 6.9% with a 4.4% margin of error downwards. XEPI-C, for which measurements were only made for a buffer size of 50, shows some level of improvement. However, from the data in this table, these improvements are not statistically significant. Figure 7 shows the CDF for each of the protocols with the same configuration (50m range, 50 message buffer size, and 4 hops). XEPI-C shows slightly better performance than XEPI alone, but not to a significant extent. This improvement in performance, however, should increase as the density of coverage increases, and individual nodes have more neighbors. XEPI-H shows abysmally worse performance than all other versions of the protocol. A more detailed evaluation of XEPI-C and XEPI-H will appear later in this chapter.

Table 3: Buffer Utilization Characteristics of EPI, XEPI, at Different Buffer Sizes with 50m Transmission Range and Maximum Hop Count of 4 hops

<table>
<thead>
<tr>
<th>Buffer</th>
<th>EPI</th>
<th>XEPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The next column in each section is the average latency of the delivered messages, in seconds. Unfortunately the variance of this value is too large for it to be more than an indicator of performance improvement, but there is a trend in which the average message latency is lower when the buffer size is low. This is confirmed by the CDFs in Figure 5 and Figure 6, in which the curve rises much more steeply for XEPI than for EPI. This does not necessarily indicate that messages are delivered faster, but rather that more messages are delivered in a short amount of time – that the frequency of the lower latencies is higher. So the use of ACKs and broadcast in XEPI provides for more reliability in terms of message delivery within a reasonable threshold of time.

Another indicator of resource consumption is the buffer utilization, effectively the amount of memory resources consumed in order to deliver each message. This breakdown is taken from [Vahdat] and calculated using a modified version of their statistical analysis script. Messages are considered ‘dead’ or ‘live’ depending on whether they still exist in the buffers of any at the end of the simulation. The next column, ‘buffers,’ indicates the average number of buffers occupied at any given point of time during the lifetime of the message, which is indicated in the third column. The lifetime of the ‘live’ messages is not indicated, since the value would be artificially terminated by the end of the simulation. Note the difference between ‘dead’ and ‘live’ vs. delivered and undelivered. Delivered messages may be dead or live, depending on whether they still exist in a buffer or not. In XEPI, ACKs should introduce a correlation between delivered and dead messages – as garbage collection begun on delivered messages will accelerate the removal of all copies of the message from the buffers of the intermediate ‘carrier’ nodes.

What makes this particular set of data especially interesting is the clear benefit provided when the buffer size is larger, with almost twice as many ‘dead’ messages for XEPI when the buffer size is 200, and almost four times as many when the buffer size is 500. In addition, in the cases where EPI does almost no space conservation,
and every message sent still lives in at least one buffer, XEPI is able to provide some level of useful garbage collection. For a buffer size of 1000 messages, and average of 71.3 messages are removed from all buffers, each of which occupies 3.8 buffers over its 943.7 second lifetime. In EPI, all of these messages remain in the buffer, with an average presence in 28.4 buffers. This amounts to a savings of 1753.98 KB (71.3 messages x 1 kb x 24.6 fewer buffers) of space within the system over the lifetime of the ‘dead’ messages alone.

It is also interesting to note that, despite the use of ACKs, there are no ‘dead’ messages when the buffer size is 2000, although the average buffer occupation rate is slightly lower than for EPI. This happens because each node eventually contains all of the messages from the system – the only thing that may cause them to remove a given message from the buffer is the receipt of an ACK. However, as can be seen in Table 4, there is only an average of 17 garbage collections per message, which means that each message will only be removed from around 17 buffers, leaving the message intact inside all of the other nodes, and thus leaving the message ‘live.’ From this, we know that ACKs alone are not sufficient to clear messages from buffers – they merely provide a better decision in terms of which messages should be kept and which should be removed, based on delivery status. Other heuristics for buffer queue management will serve to cycle messages out of buffers over time.

Furthermore, due to the small size of the ACK buffer, 200 ACKs, old acknowledgements are cycled more quickly than they can be removed from message buffers. So, although a message may be garbage collected, it is replaced quickly - the following ACK exchange replaces the entire ACK buffer, causing the node to ‘forget’ that this message was just garbage collected, and resulting in a new message request the next time it sees it. A second, brief, look at the statistics for this particular scenario (2000 message buffer), shows that there indeed may be some level of traffic churning as a result of the garbage collections. Only 99,000 messages need to be stored in order for all of the message buffers to hold all of the message in the simulation, yet we record 125,337 messages being stored, and 30,118 messages garbage collected. We may surmise that the ACK buffer size should be configured such that it can hold ACKs for each ‘live’ delivered message. Further experimentation with XEPI parameters would reveal the exact relationship between

<table>
<thead>
<tr>
<th>Buffer Size</th>
<th>Average ACKs sent</th>
<th>Average Bytes Sent</th>
<th>Average Garbage Collections</th>
<th>Bytes Garbage Collected</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>30.8 ± 10.4</td>
<td>862.4</td>
<td>0.9 ± 0.1</td>
<td>921.6</td>
<td>107%</td>
</tr>
<tr>
<td>20</td>
<td>39.6 ± 11.3</td>
<td>1108.8</td>
<td>1.5 ± 0.1</td>
<td>1536.0</td>
<td>139%</td>
</tr>
<tr>
<td>50</td>
<td>48.6 ± 14.0</td>
<td>1360.8</td>
<td>3.3 ± 0.3</td>
<td>3379.2</td>
<td>248%</td>
</tr>
<tr>
<td>100</td>
<td>52.0 ± 15.9</td>
<td>1456.0</td>
<td>5.7 ± 0.7</td>
<td>5836.8</td>
<td>401%</td>
</tr>
<tr>
<td>200</td>
<td>51.0 ± 15.3</td>
<td>1428.0</td>
<td>9.3 ± 1.3</td>
<td>9523.2</td>
<td>667%</td>
</tr>
<tr>
<td>500</td>
<td>49.1 ± 14.6</td>
<td>1374.8</td>
<td>14.0 ± 2.2</td>
<td>14336.0</td>
<td>1043%</td>
</tr>
<tr>
<td>1000</td>
<td>49.8 ± 14.6</td>
<td>1394.4</td>
<td>15.9 ± 2.5</td>
<td>16281.6</td>
<td>1168%</td>
</tr>
<tr>
<td>2000</td>
<td>50.3 ± 15.1</td>
<td>1408.4</td>
<td>17.0 ± 2.7</td>
<td>17408.0</td>
<td>1236%</td>
</tr>
</tbody>
</table>
configuration parameters, and a study of the steady-state characteristics of EPI would give us a better idea of how many 'live' delivered messages will be in the system at any given point in time.

Table 4 shows another cost-benefit analysis with respect to ACKs. Assuming that each ACK packet is 28 bytes long, we can calculate the amount of ACK traffic generated per message. For most cases, this is approximately 1400 bytes, a bit more than the size of a single message. We also know that the message size is 1KB in sizes, so each message in the buffer occupies about 1KB in space. Aside from the potential benefit garnered from not sending already-delivered messages, we have an immediate benefit of the garbage collection of the space occupied by these messages. By dividing the space previously occupied by the messages by the number of bytes sent to clear the messages, we have a benefit ratio. For smaller buffer sizes, the benefit is usually small, since most messages cycle through quickly due to buffer overflow. However, for larger buffer sizes, the benefit is a factor of 12 times gain for each byte sent. This benefit can be further multiplied if we modify the algorithm to allow aggregated ACKs, reducing the average space required to send an ACK to 4 bytes rather than 28 bytes.

![Figure 7: Comparison of Delivery Latencies for range = 50m, buffer size = 50, and max hops = 4, using Different Protocols](image)

**6.2 Steady State Random Waypoint**

Thus far, we have presented several scenarios in which we have evaluated and compared EPI and XEPI. In the interests of producing a useful benchmark, we chose to use the same mobility model used in the original research, the Random Waypoint mobility model, to define the movements of the nodes. However, over time, Random Waypoint settles into a steady state distribution with quite different characteristics from the initial random distribution. This means that results garnered from the beginning of the simulation will differ in characteristics from results garnered over a longer running simulation. As we also mentioned before, possible
Routing in Partially Connected Wireless Ad Hoc Networks

Further research would include evaluating the steady-state performance of EPI and XEPI. As a means of establishing initial research in this area, and as a means of resolving the issues behind the fact that the above statistics ignore the initialization discrepancy of the Random Waypoint mobility model. We now present a comparison of the characteristics of EPI and XEPI within both Random Waypoint and Steady State Random Waypoint.

Table 5: Comparison of Random Waypoint and Steady State Random Waypoint in EPI, XEPI, and XEPI-C, when range = 50m, buffer size = 50, and maximum hop count is 4

<table>
<thead>
<tr>
<th>Mobility Model</th>
<th>randway</th>
<th>ss</th>
<th>randway</th>
<th>ss</th>
<th>randway</th>
<th>ss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delivery Rate</td>
<td>79.5% ± 5.3</td>
<td>55.2% ± 10.9</td>
<td>83.7% ± 5.4</td>
<td>59.6% ± 12.1</td>
<td>84.4% ± 4.3</td>
<td>59.9% ± 14.1</td>
</tr>
<tr>
<td>Latency (s)</td>
<td>Avg. 246.9 ± 6.9</td>
<td>518.6 ± 112.8</td>
<td>239.4 ± 12.8</td>
<td>540.2 ± 142.0</td>
<td>232.8 ± 29.6</td>
<td>542.0 ± 151.0</td>
</tr>
<tr>
<td></td>
<td>Max 3487.8</td>
<td>3777.52</td>
<td>3168.56</td>
<td>3610.5</td>
<td>3793.55</td>
<td>3762.32</td>
</tr>
<tr>
<td>Hops</td>
<td>Avg 3.4 ± 0.1</td>
<td>3.4 ± 0.1</td>
<td>3.4 ± 0.1</td>
<td>3.4 ± 0.0</td>
<td>3.4 ± 0.1</td>
<td>3.4 ± 0.0</td>
</tr>
<tr>
<td></td>
<td>Max 5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

It is apparent that the performance is worse when using the ss mobility model, with a 20% decrease in delivery rate, and a doubling of the average delivery latency. This may be due to a change in the number of useful pair-wise exchanges. In the initial configuration of Random Waypoint, just a random distribution, nodes are evenly spread throughout the network. This even density has an impact on performance in that the initial exchanges happen more quickly, since no nodes are more out of range than others. Once the configuration settles into a stationary distribution, however, the nodes move towards the center, resulting in many nodes being at higher density, but nodes on the edge having lower connectivity. [Navidi]
Figure 8 and Figure 9, CDF graphs of delivery rates within Steady-State Random Waypoint, in conjunction with Figure 5 and Figure 6, graphs of the same in Random Waypoint, further highlight the differences in the characteristics between the two models, while also showing some key similarities. The same trend, in which message delivery rate is increased with XEPI, especially at lower buffer sizes, exists in both models. Furthermore, increased buffer size results in higher message delivery.
However, there is a key difference in the shape of the curve. Where Random Waypoint rises sharply in the first 200 seconds, and then flattens out, Steady State rises slowly, and flattens out as it approaches its maximum delivery rate for that model. However, a look at the CDF for ss over 4000 seconds instead of 750 s shows that the curve is actually similar in shape, but merely ‘stretched’ out, for a longer latency time before the message is delivered to its final destination.
Chapter 7: Application

7.1 Implementation

As a proof of concept for the eXtended Epidemic routing algorithm a wireless message program similar to an instant messenger was created. The reason that this messenger unlike messengers such as those created by Microsoft, AOL, or GAIM, is that this messenger firstly cannot be called an instant messenger. Messages using this messenger can be stored on multiple hosts as stated earlier in the description of the eXtended Epidemic algorithm. These messages will then attempt to make their way to their intended destination and can be buffered for long periods of time. As we are dealing with partially connected networks some messages need to be carried by nodes towards the intended destination. All of tasks which the algorithm performs have been recreated at the application level in a Java program. In this section the requirements and design of the Ad-Hoc Messenger will be discussed.

7.2 General Overview of System Functionality

The reason it was decided that a messenger application would be created was that it is a good example of asynchronous communication. This is where a message can be delivered and it is not necessary for a message to return to the host system. While our version of epidemic routing does use acknowledgements these are not for retransmissions, but instead to help clear buffer space in nodes around the sender and receiver. Retransmissions are not needed in this system as the reliability is provided by fact that multiple copies of the same message are within the system, and should one copy be corrupted other copies which are not corrupted will still arrive at the destination. This does however mean that it is necessary to keep a buffer of the unique message sequence numbers so that duplicate messages are not displayed at the destination.

One of the considerations that had to be thought of from the start was that this application would be intended for use on wireless networks. Now wireless networks have different characteristics to wired networks. And this is one of the main considerations that need to be taken into account when programming an application for this type of scenario. Firstly the most obvious element of wireless networks is that connectivity can not be assured as nodes might move out of range, or just loose power. As such, before starting any design work, a similar project done using the Microsoft .NET framework [ref#121] was looked at. While this project related to a similar field, it had a lot of significant differences to what was being planned for this implementation. Two of the main differences which will be looked at, are that they assume constant connectivity between hosts. In doing so they are also able to guarantee instant delivery.
A second paper which looks at the use of MoM (Message orientated Middleware), and in particular JMS (Java Message Service) as a middleware for creating software for MANETs was also studied. The paper [22] discussed the creation of an Instant Messenger which uses the ODMRP (On Demand Multicast Routing Protocol) while using JMS as a middleware. This implementation again is different from the Messenger designed for the eXtended Epidemic routing algorithm as it relies on a connected network to ensure message delivery.

One of the preliminary decisions which relates to the eXtended epidemic routing algorithm is that we are dealing with partially connected networks. As such it is necessary to buffer messages and use the UDP protocol, rather than the reliable point to point TCP protocol. The messages which are intended for nodes which may or may not be in indirect contact need to be buffered in our situation. This is a key feature of epidemic routing, and so has to be implemented in the application. As it is not always possible to deliver a message immediately we therefore cannot, or even want to, claim to have an instant messenger.

A second decision that had to be made was that if the messages need to be passed around the network, where a single message traverses multiple nodes, how would a node identify which message was destined for it. For this a single unique identifier would be attributed to each host. As their IP address would not necessarily be the best choice, due to possible IP conflicts as it might be possible to have the same IP address used by more than one host as long as they were in separate clusters.

There are a few problems with using Java to create applications for MANETs. One issue is that you are not able to have direct access to the WiFi card, and so cannot set the card to be in promiscuous mode. So instead it was decided to broadcast the packets to the broadcast address for the subnet (i.e. xxx.xxx.255.255). This simulates the broadcast properties of wireless communications and at the same time ensures that all hosts within broadcast range should be able to pick it up and pass the packet up to the application level.

While it is not common to see a routing algorithm being implemented at the application level, due to the reduction in performance, this is used to the advantage of the application as the routing algorithm now is able to use application level knowledge in making routing decisions. This also allows for future modification to be made where more application level knowledge is employed. Not all of the modifications suggested in this paper to the epidemic routing protocol have been implemented however. One such example is the using of history data to predict message flows as mentioned earlier.

In order to gain knowledge of nodes' surroundings, every node transmits alive messages every 20 seconds. These messages are passed on for a set number of hops. This allows nodes to be able to gain application level knowledge of their surroundings, as well as giving the user of the application an idea as to who is online. While nodes that have not transmitted an alive message for 2 minutes are set as being offline it is still possible to transmit messages to them. The reason for this is two fold. Firstly it is possible that the node is still online however has moved to a new cluster, in which case transmitting the message could still result in delivery.
Secondly it is possible that the node has gone offline temporarily and should it come back online and the message is still held in a buffer it will be delivered as well.

While buffering all these messages does use some system resources, the buffers are not infinite and so messages will not hang around the system for ever. Each buffer uses a FIFO queue to ensure messages that have been received but never delivered do not remain in the system indefinitely. Messages in the buffers are also removed when the corresponding acknowledgement is received. So if there is only a light amount of traffic it is conceivable that messages will stay around for some time before either the node switches off, or the traffic load increase.

Messages do not however get sent round endlessly. Each message contains a counter which is incremented every time it gets transmitted. When this counter reaches the allowed maximum hop count it will no longer be forwarded unless the destination node becomes the neighbour. This ensures that messages do not go round the network indefinitely, and so taking up valuable resources. The maximum number of hops is a set value that can be altered within the code, but not the application while running. The reason behind this decision is that it would be possible for users to set the value to an almost infinite number of hops. This would then ruin the service for the other users which would be using the shared network sensibly.

For auditing purposes each messages also keeps track of every node that it traverses. Once again the reason for this is two fold; security, and intelligent application level routing decisions. Keeping an audit trail allows for greater security should you wish to not trust a node you would then be able to see if your message had hopped along that particular node. Intelligent routing decisions could be made using this knowledge. It would be possible to send a message towards a given direction by knowing who you got it from and likewise they know who they got it from. This feature however was also not introduced into this application, but could be inserted during future work.

### 7.3 Overview of System Components

The programming of the application was done using Java 1.3, which allowed for object-orientated analysis and design. The program was broken up logically into the various components. This also allows for future re-usability of certain classes, such as the GUI classes which are able to act as a front to any messenger system. This section will look at the various components of the application in more detail as well as explaining the decisions behind their design. The class diagram showing the relationships between the different classes in the application can be seen in Figure fig#1.
IMApplication Class
This class deals with the starting of the program, as it contains the main method which prepares and then launches the applications main window. Apart from launching the main window this class is also called upon at later times to prepare and launch conversation windows.

MainWindow Class
Once called this class creates the GUI which can be seen in Figure fig#2. The purpose of the MainWindow class is to provide an easy to use front end to the messaging application. This front end allows the user to start and stop the client from accessing and listening to the network for traffic, as well as allowing the user to select a 'buddy' from the list with which to have a conversation. The buddy list which is visible on the GUI is controlled by the UserTableModel class.

StartupUserDialog Class
This class is called when the application is first run. It is a GUI class that allows the user to input two key values. Their unique identifier, in this case their e-mail address
and the display name they wish other people to see when they are writing. After the initial setup data has been entered this class is no longer used.

**ConversationWindow Class**
This GUI class deals entirely with user conversations. It is used to display the conversation between two users, as well as get the users’ text which is passed on to the IMClient class to be sent to the other party. The class is called through the IMApplication class, where it is prepared and then created. There can be multiple instances of this class in the program. This allows for the users to talk to as many people as they want to at the same time.

**IMClient Class**
This class is the heart of the application, and where the routing is preformed. The IMClient class once launched is a thread, and waits for the start command from the MainWindow. Once the class is started it deals with all the network related function, including sending a message, receiving a message, and then dealing with the message once it has been received.

This class contains the eXtended Epidemic algorithm. As such it deals with the following different types of messages; alive messages, data messages, acknowledgement messages, and hash table exchange messages which include both buffered acknowledgements, and buffered messages.

Before the messages can be transmitted this class also creates the messages from the data that needs to be sent. It does this by getting the various pieces of data together and making a new Message object containing the relevant pieces.

While the thread is running it periodically transmits alive messages to inform other nodes in the cluster of its existence. However it also every 120 seconds goes through the list of all the ‘buddies’ to see if any of their states have changed. So for example if no alive message has been received from a node for a while it will set that node as being offline. This means that it cannot guarantee that the message will ever get to the intended recipient.

The algorithm also calls a few methods in this class that deal with the storing of messages and acknowledgements into the buffers that it will attempt to send to other nodes, if and when they come into direct contact with this node.

**UserTableModel Class**
This class deals exclusively with the displaying of information on to the JTable which is found in the MainWindow class. The main role of this class has to do with updating and adding new ‘buddies’ which have either just appeared and so they will be displayed, or their status has changed and they will be updated.

**UserInfo Class**
This is a record class, which is used to store information of a single user. Each user when they are discovered gets all their information stored in this class. This allows
for easy future modification should more data need to be stored about a user for increased application level routing knowledge.

**Message Class**

The Message class is similar to the UserInfo class in that it is the record which holds all the message data which needs to be transmitted. This again can be updated in the future to allow for more information to be passed between nodes. The format the message takes is as follows:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 ...
+-----------------------------------------------+
0|   |   |   |   | T |   |   |
1|V| IEEE | Seq | | P | IEEE | Data | IEEE |
2|E| MAC | | Num | | A | MAC | | MAC |
3|R| ADDRESS | | | | ADDRESS | | | mixed ADDRESS |
4|S| | | | L | | length| TRAIL |
5|I| (Nexthop) | | | | | | psble | (Source & | |
6|0| X | X | | 1A | 0 | | all passed |
7|N| | | | P | | | | node(s)
+-----------------------------------------------+
<-------HEADER-(23 Bytes)----------------------<-PAYLOAD<=(1000 Bytes)-->
```

As can be seen from the packet format, the first byte is a version number. The version number relates to the type of packet. If it is a ‘1’ it is an alive packet, and ‘2’ indicates that it is a message packet, and that continues for the different message needs. The sequence number is generated in the IMClient class and to try and make it as unique as possible the last three digits from the IP address are used as a seed. And that number is then multiplied by the maximum buffer size plus a bit extra. And when a node has gone to its top limit for the sequence number then it rolls over and begins with the early numbers again. This ensures that there are no two messages with the same sequence in the same cluster of wireless hosts. At the rear of the packet the audit trail is stored.

**NetworkTools Class**

The sole role of this class is to convert Java objects into bytes so that they can be transmitted across the network. Likewise it performs the reverse of this operation when a new message is received from the network, where it then converts from bytes to an object. This method can be used to convert any form of object to allow for transmission. As such this class is not just re-usable for this application, but for any network based application.

**7.4 The Graphical User Interfaces**

All the user interfaces were designed to be as easy and intuitive to use as possible. This will allow for rapid learning of the application by new users. Following is a brief description of the different interfaces within the application.
The Main Window

The following figure shows the design of the main window. In Figure fig#2 the client has is still offline, and after the ‘Go Online’ button has been activated the application automatically gets the user’s IP address, as well as the broadcast IP address from the subnet mask. In Figure fig#3 the client has been started and is online. Now the user has the option to ‘Go Offline’, or chat with one of the ‘buddies’ displayed in the table. In the table the user is shown some of the information on the buddies, such as; their system name / IP address, whether the user is a direct neighbour, whether the Messenger has heard from the neighbour in the last 2 minutes, and finally the unique username of the user. Should the user wish to chat with one of the users, it is possible to select a user from the table and press the ‘Start Chat’ button.

![Figure fig#2 - Main window prior to client start](image)

![Figure fig#3 - Main window after client start](image)

The User Setup Dialog

The user setup dialog as seen in Figure fig#4 allows the user to give the application their details. The two items that it asks of the user are a unique user id, and the
display name. It is recommended to use the users e-mail address as the unique identifier as they are unique by definition, and so allowing the transmitted messages to be tied to their ID rather than to an ID that might be used by more than one user. The display name is used in the conversation window as opposed to having the entire user ID displayed each time.

The Conversation Window
The conversation window shown in Figure fig#5 displays to which user you are messaging, as well as the conversation that has taken place between the two users. The send button at the bottom causes the message in bottom box to be sent to the IMClient where it is able to packaged and sent off to the destination. The conversation window can also be started directly from the IMClient via the IMApplication class if a message arrives for the user. There can be as many instances of this window open at one time to different users.
Chapter 8: Conclusions, Evaluation and Further Work

8.1 Conclusion and Evaluation

In the original epidemic routing a lot of messages keep floating around the network even after they have been delivered to the final destination. Those messages occupy extra buffer space in the nodes competing for it with undelivered messages and create additional overhead on the network as nodes keep exchanging those messages with each other until they are replaced with newer messages.

We show that using of acknowledgements for garbage collections can significantly optimise the buffer space utilisation. The amount of dead messages in the network is optimal if the size of the acknowledgment buffer is comparable to the size of the message buffer. Better buffer utilisation led to a clear performance gains especially when the size of the buffer is limited, with larger gains with smaller buffer sizes. This indicates that individual nodes have more space for the more relevant messages.

The usage of acknowledgements for garbage collection does not give much performance advantage if the buffer space is unlimited. However, acknowledgements may still be very useful to limit the total amount of traffic in the network. This is one of the areas for future research.

Broadcast optimisation did not had much impact on the performance of the epidemic routing. We can attribute that to a low density of the nodes in our mobility scenes which does not allow to utilise the advantages of broadcast instead of unicast. Analyzing the impact of broadcast in dense networks is another area for future research.

We have also researched the impact of different mobility models on the performance of epidemic protocols. The results of our simulations show that the performance of epidemic protocols with a stable state random waypoint mobility model is significantly worse, with 20% decrease in delivery rate, and a double increase in latency. This is explained by the fact that the initial distribution of nodes in the simulation area is not representative of the steady state distribution of random waypoint mobility model.

The proposed history optimisation showed abysmally poor results. This demonstrates how a sub-optimal queuing discipline can dramatically affect the performance of epidemic routing protocol. We suspect that the use of history in this way disallowed all intergroup communication.

All in all, the project was a success. We were able to propose several extensions to epidemic routing, and show that they improved performance within the system. We also created a new mobility model, potentially useful for further research in ad hoc networks in general. In addition, we have an application that demonstrates a practical use of the work we have implemented. Limits of resources, such as linux
machines, and disk space, as well as time, prevented us from doing much of the further research we would like to have done, but the work we were able to do is still significant, and have resulted in demonstrated improvements.

Copies of our code, generated statistics, and this document are available on CD.

8.2 Future Work

Epidemic Routing Extensions

While working on optimisations of epidemic protocol we came up with a number of ideas which can be a subject of future research.

- Using ack vector rather than acks in separate packets. Currently acknowledgments spread in the same manner as messages and each acknowledgment takes up a whole packet which is at least 28 bytes (including the IP header). It would more efficient to exchange vectors of acknowledged message IDs rather than acknowledgements themselves.

- Message prioritization. In a fast moving environment there may be not enough time to do a message exchange. We suggest to prioritize the list of messages to be exchanged by their broadcast counter values. The higher the counter value the more probability that this message can be received from someone else. Therefore the higher the counter value, the lower the message priority in message exchange.

- History heuristics. We suggest to use history information not only for queing discipline but for making decisions whether to accept this message at all. The history algorithm should take into account not only node contacts, but also the source addresses of the packets in the buffer.

- It is possible that defining the order in which nodes send messages may increase the contention of the channel in dense networks. It is possible to define the order on the basis of the lowest IP address of the neighbours.

Data Analysis

- Analyze the characteristic performance of XEPI, XEPI-C, and XEPI using different parameters, as specified in the user manual. Determine optimal parameter settings, and what conditions affect the parameter settings.

- Analyze the steady-state characteristics of Epidemic Routing.

- Observe the characteristics of XEPI-C and broadcast in dense networks.

- Develop and study appropriate mobility models.

Application

There are quite a few areas of future work feasible, as the Ad-hoc messenger currently is implementing the eXtended Epidemic routing without using some of the application level knowledge which is available. There are also a lot of possible modifications to the program which would allow it do have more functionality. And finally a key factor which is not implemented here but would be a very interesting research area would be to add security to the whole model.
Using application level knowledge it is possible for the Messenger to make as it were ‘intelligent’ decisions. These would all relate to the routing of messages through the wireless ad-hoc network. This can include intelligent back tracking, where the audit trail can be used to send a message back quicker without using as many other resources until such a point is reached where the trail can no longer be followed. However assuming the node might still be in that general direction, using the epidemic feature later on would save resources being used which flow in the opposite direction from the actual destination.

More steps in a similar direction would be to allow the application to make decisions on the most effective routing protocol to use and then implementing a hot-swappable routing system where different protocols could be used interchangeably. This could add to overall efficiency, especially if you are dealing with nodes in a connected network it would be possible to a multitude of efficient routing protocols. This means that the application would need to assume scenarios exist for which different routing protocols are more effective.

Adding more functionality to the application would include items such as different text colours, saving the conversation, and other minor add-ons. However the most interesting of these add-ons would be adding security to the whole application. This would include making it impossible to read other users messages in transit, and being able to authenticate who you are so not everybody is able to claim they are someone else.
Appendix A: Survey of Mobility Models for Epidemic Ad Hoc Routing

[Revision History]
* 30 June - Initial Revision (mh)

[Abstract]
This document seeks to examine the requirements for designing sufficient models for evaluating ad hoc routing algorithms, particularly epidemic algorithms and our planned improvements on the algorithm. Currently used models will be evaluated, and new models or suggestions to improve existing models will be proposed. This will also suggest parameters for existing models that we should use to evaluate our routing algorithm.

(Network Modeling)
A model is a simplified version of a complex system. As we learned in Z09, Network Modelling and Monitoring, in order to build a model, we need to observe a set of steps known as the models cycle. For your reviewing enjoyment, and to provide context for this paper, I will summarize his notes here.

What is a model?
* A simplified representation or description of a system or complex entity, especially one designed to facilitate calculations and predictions
* We recognize our world as a system of objects and attributes
  - Objective: attributes you can "measure" and put a value to
    - Real: directly measurable (e.g. time, distance)
    - Imaginary: composite attributes created from real (e.g. speed = d/t)
  - Subjective: attributes for which we have no agreed means of measuring
* Models vs. Instances vs. Copies
  - copies are best effort, but in models one or more attributes have been deliberately simplified
  - instances can not be distinguished from each other, model may be synonymous with type, but in this case not a simplification

Models Cycle
Define Goals
Define Model Limitations
Select Attributes - choose crucial attributes to represent system
Select Metrics - a small number are chosen to evaluate how close the model comes to fulfilling expectations
Select Factors - independent variables we want to investigate
Select Evaluation Method - what technique we can use to perform analysis
Validate Model - verify consistency of model specifications
  * Functional - approximations accurate in entire range of interest
  * Distributional - well-behaved probability distributions may not really be representative
  * Independence - independence of variables may be result of model design
  * Aggregation - esp. temporal aggregation in discrete time and discrete event studies
  * Stationarity - be aware of impact of assumption that one or more attributes are constant during lifetime of model
Perform Experiments
Reduce and Evaluate Sampled Data
Propose Conclusion
Propose Experiments to Evaluate Conclusions

Models Cycle Application
Goals:
* Realistically evaluate an ad hoc routing algorithm
* Perform comparative analysis with existing algorithms
* Produce a model that approximates theoretical usage
Model Limitations:
* No realistic data on ad hoc network structure, mobility, and usage
* Complex models are harder to validate and evaluate, and implement
* 3D ad hoc models are more difficult due to directional nature of many wireless transmissions (mostly planar, with a little horizontal access).

Routing algorithm must be evaluated in a simulated environment, due to lack of resources and the impossibility of reproducing hundreds of mobile nodes and measuring the data associated with them.

Attributes:

* Motion model – how do the mobile hosts move?
* Motion graphs – where can they move?
* Resource consumption – buffer space, communication overhead, etc.
* Density – how many hosts, how large a field, and what shape?
* Connections – 802.11 emulation, range of hosts, errors, fading, etc.

Metrics (based on Epidemic Routing metrics)

* This one is a little dubious, due to lack of normative data for comparison.

* Comparison to previous work – reusing models or elements thereof

Variables (based on Epidemic Routing metrics)

* Range of mobile nodes {10, 25, 50, 100, 250} m
* Mobility model and associated parameters (see below)
* Buffer size {10, 20, 50, 100, 200, 500, 1000, 2000}
* Message hop limit {1, 2, 3, 4, 8} hops

Factors

* Delivery Rate (compared to baseline, as well)
* Latency (avg, max)
* Hops (avg, max)
* Coverage Floor
* Buffer Utilization (dead/live)
* Lifetime of Message (avg, max)
* Memory and resources required for given delivery rate and latency

Evaluation Method

* Analysis of trace file, producing comparison lines and graphs
* Delivery rates at various radio ranges (Percent of messages delivered vs Message delivery latency)
* Characteristics of Routing as a function of transmission range, also as set against baseline rate of delivery using traditional ad hoc routing – AODV – in same scenario (delivery rate, latency, hops)
* Delivery rates at various hop counts (percentage of messages delivered vs message delivery latency)
* Delivery rates at various buffer capacities with 4 hop limit
* Resource consumption characteristics (50 m range, 4 hops, variable buffer size)

* Comparison of all of the above in different mobility scenarios

Validate Model

* We realize that there are inherent limitations in simulating the ad hoc environment, and essential simplifications.
* Testing a variety of mobility models gives us the ability to determine the scenarios in which this algorithm will be most viable
* Prior research gives us a benchmark for comparison, and a standing point for what validity can be had.
* Our chosen models should be a validated improvement upon currently used models (mostly Random Walk) and not unfounded.
* We should provide clear reasoning for each of our modifications, as well as an analysis of why they are logical improvements.

Perform experiments

* ns - widely used network simulator
* emulation - be sure to validate implementation of emulator

Reduce and evaluate sampled data

* Plot the data in a number of different co-ordinate systems with linear, logarithmic, or other scales. When they form a line, you have a relationship. (von Karman)
* Explain any outliers, or results are invalid
* Run tests until variance drops below a pre-specified level

Propose conclusion

Propose experiments to evaluate conclusion

[Current Research]

This section will summarize existing models in use, as well as some of their justifications. In general, sufficient justification is not provided, and the models chosen are fairly simple, except to show that sufficient time has been spent in the model to reduce variance to a reasonable level.

Epidemic Routing

* 50 nodes
* Rectangular area, 1500x300m, to provide sufficient distance between
Routing in Partially Connected Wireless Ad Hoc Networks

- Random walk mobility model
  - each mn picks a point and speed, moves there, and repeats the process
  - 0-20m/s (avg 10 m/s)
- Messages are 1 kb in length
- 45 nodes send messages to 44 other nodes
  - total 1980 messages
  - one message sent per second (all messages delivered after 1980 s)
  - 2000 slot buffer is essentially infinite buffer space
- vary range, buffer size, max hop count
- Note: would expect steady state to arise, given regular injection of new messages, but terminate sending in order to observe behavior of a particular set of messages.

Chen and Murphy 2001
- The algorithm is described in detail (utility models) but actual experiments do not seem to have been conducted, so no particular mobility model has been suggested

Gossip (Haas)
The paper on gossip-based routing evaluates the algorithm in several different scenarios, incorporating simple ones to illustrate the idea and potential problems, and showing one more complex scenario.

I. 1000 nodes in a 20x50 column grid
- no mobility
- k (connectivity) = 4
- not typical of topology we expect in ad hoc networks

II. 1000 nodes randomly distributed in 7500x3000 m region
- range of 250 m
- k = 8
- no wrap-around mesh, since expect real networks to have boundaries

III. 1200 nodes in same configuration as II.
- k = 10

IV. 1,000,000 nodes in 1000x1000m region
- source in the center of row 10, to avoid boundary effects
- specifically to evaluate theta-k(p), the probability that a node receives and forwards a message given that the message does not die out

V. 150 nodes in a 3300m x 600m region
- mobility determined by random waypoint model
- provides worst-case estimate of the performance of gossiping
- also tried with more 'square' layout - 1650m x 1200 m
- 30 connections
- 525 s running time
- 2 packets/second
- 0-20 m/s, average 10 m/s
- variable pause time (0 pause time results in increased load, but lower end-to-end delay)

Greek Paper
- Mobility of nodes exchanging data is determined by the protocol
  - Snake: Maintain pairwise-connectivity by following each other in chain
  - Runners: Independent Random Walk
- For non-carrier nodes:
  - move to random adjacent point on graph
  - generate new message with probability p = 0.01
    - new message every 100 rounds
    - random destination host
    - assumption that there is only non-continuous exchange of messages
    - experiment ends when 100,000 messages have been transmitted and delivered

* Graphs:
  - Random - considered natural starting point for experimentation
    - sample edges of a complete graph with probability p > connectivity thresh
      - n = { 1600, 3200, 6400 }
      - k = [3, 45]
  - 2D Grid - simplest model of motion sqrt(n) x sqrt(n)
    - n = {400, 1600}, over different values of k
  - 3D Grid - model motion in 3D space
    - n = {512, 1000, 1280}
    - k = [3, 45]
  - Bipartite multi-stage graphs
    - essentially layers of chess boards connected randomly by edges
    - log(n) stages
    - model movements of hosts that need to pass through certain places or regions, with a different probably of movement between regions
    - n = { 1600, 3200, 6400 }
Routing in Partially Connected Wireless Ad Hoc Networks

- \( k = [3, 45] \)
- 7, 8, 9 stages
- Two-level motion graphs
  - most mobile users travel along favorite routes (highways, etc), and there are more congested areas in which there is a higher volume of traffic
  - subgraphs, interconnected by small number of paths
  - hosts are forced to move within subgraph or along paths connecting them
  - vary diameter \( c \) of congested areas, and number \( f \) of subgraphs
  - each destination chosen at random

This particular study is interesting for the multi-stage graphs and the two-level motion graphs. All other studies have used Random Walk, or Random Waypoint, models that are easy to study but not practically representative. An attempt is made to approximate features of real motion, by constraining the paths and simulating congestion. We would like to consider these models further for evaluation of our routing protocol. In particular, the ideas of favorite routes or subgraphs connected by less-traveled paths seems ideally suited to the scenario described by epidemic routing.

[Proposed Models]

Message Delivery Model
I find the message delivery models used in the papers to be problematic. In epidemic, each node sends a message to all other nodes, at a rate of 1 per second. Every message is 1kb in size, and sending of messages terminates after a time, to allow all messages to be processed. No attempt is made to measure the steady state performance of the system - just the average over 1980 seconds.

In the greek paper, each node sends with a probability \( p \), and the destination is picked out at random. These are both unrealistic traffic patterns - communications will not likely be that symmetric, nor that short in duration. Although these are valid models for evaluating the feasibility of message delivery, they are invalid for measuring load, which is our primary concern for the moment.

There are three main scenarios suggested for ad hoc networks, all of which would require different models. In sensor array networks, each node will probably be broadcasting all of its info to all the other nodes periodically. In military networks, a lot of the data will be broadcast, with a very few number of private or group messages, between officers or from the officers to their units. In day-to-day ad hoc networks, we expect individual nodes to establish contact with one or more entities, exchange a number of messages over a period of time. Thus message exchange should continue, even if the entities are moving apart from each other.

As a disclaimer, this scenario is merely an improvement on the existing message delivery scenarios, and not necessarily a picture of reality. To emulate this, I propose the following model:

for each node:
1) Pick a probability of transmission initiation (<.5, ~.01)
2) Pick the time until the next transmission, according to a stationary distribution (mean is .01, not uniform, provide justification for whatever distribution we choose)
3) Pick transmission duration and message sizes, also according to some reasonable distribution.
4) Schedule series of messages
5) Schedule other transmissions similarly, overlapping transmissions are allowed. (ie, go to 2.)

Random Waypoint
This model will be used as a benchmark for comparison against epidemic routing. However, in the means used in the original paper, we find that there are several problems and inaccuracies inherent to using this model, which are not addressed by the writers of the paper. Namely, after a time, mobile nodes in this model reach a steady state, and thus, the first \( n \) seconds of the simulation are invalid. Tracy Camp's paper, "Stationary Distributions for the Random Waypoint Mobility Model," asserts that a node traveling at an average of 10 m/s (+/- 10m/s) will not achieve stability for approximately 100 seconds. However, in the interests of benchmark comparison, we must use the model as presented in the paper:

* 50 nodes in 1500 x 300 m
* speed is 0-20 m/s, with an average of 10 m/s
* 1KB message
* 45 of the nodes send messages to 44 other nodes
Routing in Partially Connected Wireless Ad Hoc Networks

- 1980 messages, 1/second
- 2000 slot buffer --> infinite buffer space
- buf=2000, hops=3, range = [10, 25, 50, 100, 250 m]
- buf=[10, 20, 50, 100, 200, 500, 1000, 2000], hops=3, range=50
- buf=2000, hops=[1, 2, 3, 4, 8], range=50m

We will try both the this message model and the model described above.

Steady State Random Waypoint
In addition to the benchmark above, we will use the steady state random waypoint model, as described in the aforementioned paper. The same experiments will be repeated, for both the standard epidemic routing and the extended epidemic routing.

If this produces good results, we will duplicate this experiment with the message model as described above.

Multistage Random Waypoint
This model is proposed as a means of evaluating the merits of using history to improve performance of ad hoc epidemic routing. The previous models have a single state memory - there is no history inherent to the models because the movement is random. To create a history, while maintaining a random element we propose a model in which individual nodes have different characteristic patterns of travel:

- Nodes may stay within a certain locality, or range everywhere
- Nodes will have different typical traveling speeds
- Nodes may change pattern of travel over time

To simplify this, we will adopt the scenario most commonly used to illustrate epidemic routing. There will be three groups of nodes. The nodes in two of them will move outside of the range of the other group (100 m distance), and the third group will be the "roamers", and will range over the entire field. Previous research indicates that given a specific set of parameters in the random waypoint model, the nodes will largely stay within the same area, emulating localized movements. So, with the three groups having different parameters we will be able to emulate a "history" in which the roamers will reappear periodically and carry messages between the two other groups.
Appendix B: Mobility Models
User Guide

Part 1: Finding and Compiling the Source

All of the code for mobility models is available in the CVS repository under $CVSROOT/toilers. The originally checked in code (1.1) is the source provided by the Toilers Group. To make the executable, go into the directory appropriate for the mobility model you want to use (detailed in the next section) and type make. It should compile under any platform.

Part 2: Mobility Model Generator Parameters and usage

We use the following mobility model generators. Note that for each of these, <seed> is an additional argument, allowing the user to specify the seed to be used for the random number generator. Pass the value '0' to use the current time as the seed.

<table>
<thead>
<tr>
<th>randway/ mobgen: Random Waypoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage: mobgen &lt;number of nodes&gt;</td>
</tr>
<tr>
<td>&lt;max-x&gt; &lt;max-y&gt; &lt;end time&gt;</td>
</tr>
<tr>
<td>&lt;speed mean&gt; &lt;speed delta&gt;</td>
</tr>
<tr>
<td>&lt;pause time&gt;</td>
</tr>
<tr>
<td>&lt;pause time delta&gt;</td>
</tr>
<tr>
<td>'&lt;N' or 'G'&gt; &lt;seed&gt;</td>
</tr>
<tr>
<td>'N' implies NS2 mobility file</td>
</tr>
<tr>
<td>'G' implies gnuplot path file</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mobgen-ss/ mobgen-ss: Steady State Random Waypoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage: mobgen &lt;number of nodes&gt;</td>
</tr>
<tr>
<td>&lt;max-x&gt; &lt;max-y&gt; &lt;end time&gt;</td>
</tr>
<tr>
<td>&lt;speed mean&gt; &lt;speed delta&gt;</td>
</tr>
<tr>
<td>&lt;pause time&gt;</td>
</tr>
<tr>
<td>&lt;pause time delta&gt;</td>
</tr>
<tr>
<td>'&lt;N' or 'G'&gt; &lt;seed&gt;</td>
</tr>
<tr>
<td>'N' implies NS2 mobility file</td>
</tr>
<tr>
<td>'G' implies gnuplot path file</td>
</tr>
</tbody>
</table>
**multi-rwss/ multi-rwss:**
Multistage Random Waypoint Steady State

<table>
<thead>
<tr>
<th>Usage:</th>
<th>multi-rwss</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;number of nodes per group&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;max-x&gt; &lt;max-y&gt; &lt;end time&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;roam speed mean&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;roam speed delta&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;roam pause time&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;roam pause time delta&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;speed mean a&gt; &lt;speed delta a&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;pause time a&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;pause time delta a&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;min-x a&gt; &lt;max-x a&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;min-y a&gt; &lt;max-y a&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;speed mean b&gt; &lt;speed delta b&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;pause time b&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;pause time delta b&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;min-x b&gt; &lt;max-x b&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;min-y b&gt; &lt;max-y b&gt;</td>
</tr>
<tr>
<td></td>
<td>'&lt;N' or 'G' or 'Q'&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;seed&gt;</td>
</tr>
<tr>
<td></td>
<td>'N' implies NS2 mobility file</td>
</tr>
<tr>
<td></td>
<td>'G' implies gnuplot path file</td>
</tr>
<tr>
<td></td>
<td>'Q' implies QualNet mobility file</td>
</tr>
</tbody>
</table>

For Multistage Random Waypoint, there are significantly more parameters. You will need to specify the speed and pause time for each of the groups. Since the roamers by definition roam over the entire field, you do not need to specify their locality. However for the localized groups (a and b), you will need to specify the coordinates within which they will need to limit their movements. Another key difference is that you need to specify the number of nodes per group rather than the total number of nodes. The total number of nodes will be three times the number you specify, since there are three groups.

**Part 3: Mobility Models Used In This Paper**

To generate **randway:**
> mobgen 50 300 1500 4000 10 10 0 0 N <seed>

To generate **ss:**
> mobgen-ss 50 300 1500 4000 10 9.9999 0 0 N <seed>

To generate **multi:**
Routing in Partially Connected Wireless Ad Hoc Networks

> multi-rwss 20 300 1500 4000 10 9.9999 0 0 10 5 0 0 750 0 300 15 5 0 0 750 1500 0 300 N <seed>

To generate small:
> multi-rwss 20 600 300 $endtime 10 9.9999 0 0 10 9.9999 0 0 50 250 50 250 10 9.9999 0 0 350 550 50 250 N <seed>
Appendix C: Running the Simulation and Generating Statistics

All of the scripts used to run the simulation are in the CVS repository, under epi/analysis. Everything is automated, so you can run all of the simulations and generate the relevant statistics using the same perl script, runall.pl. The following is a listing of the options for the script. You can edit the script to turn on or off any simulations as needed.

<table>
<thead>
<tr>
<th>runall.pl</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reps=[number]</td>
<td>The number of times each set of simulations should be run.</td>
</tr>
<tr>
<td>stats=[yes</td>
<td>no]</td>
</tr>
<tr>
<td>sim=[yes</td>
<td>no]</td>
</tr>
<tr>
<td>quick=[yes</td>
<td>no]</td>
</tr>
<tr>
<td>debug=[yes</td>
<td>no]</td>
</tr>
</tbody>
</table>

Running the simulations will generate several sets of trace files, each set in a directory corresponding to the protocol in use. The name of the trace file will indicate the configuration and mobility model used to generate the trace file. The seed used to generate the mobility model is also included in the filename. Two files will be generated, a .epi file, which contains trace data as generated by the original
implementation, and a .ack file, which contains the trace records generated by our implementation. One or both of these files may also be compressed.

Generating the statistics will create two files, a variance file (var.*), and a cumulative distribution function file (.cdf). The variance file lists the stats from each individual simulation, as well as the cumulative average, variance, and standard deviation of several statistics over multiple simulations of the same configuration. The listing of the cumulative statistics appears at the bottom. The cdf file is a two column data file for use with xgraph (or any other graphing program). The graph generated will be the cumulative distribution of message latencies for that particular configuration, and will incorporate all of the data from the various simulations using different seeds for the mobility model.
Appendix D: Area of application

802.11 hot spots are new and interesting area. 802.11 is cheaper than 3G and much faster. Significant problem are hot spots. 802.11 base station has maximum 100 m of coverage that is too small for wide commercial usage.

BT, Intel and other big players have invested significant amounts of money in development of hot spots and related technology. BT already has 4000 hotspots around the UK and plans for more. Many cafés, pubs, airports and hotels offer 802.11 hotspots.

Let us concentrate on coverage problem. Each hotspot has limited radius (100 meters). It would be expensive to cover entire country with hotspots. Instead, we can use other 802.11 devices as relays. This approach will extend coverage of the hot spots using cheap and simple technique (ad hoc + epidemic) and increase number of users (cheaper and larger coverage area).

Any user using ad hoc routing protocol (TORA, AODV, DSR, etc) in fully connected manner within few hops from hot spot will be able to get good speed and quality (enough for real time or multimedia). Epidemic routing can be used to extend hot spots reach even further. Quality of data transmission will be lower and not suitable for real time apps but will be sufficient for asynchronous traffic (emails, lightweight asynchronous applications, push type information, synchronizing schedules, etc.)

802.11 can be competitive compared to 3G networks, offer better speeds, cost less and have extended coverage. Wireless users serve as part of wireless ad hoc infrastructure and help reducing the costs of network. Companies can pass this saving directly to users and offer competitive wireless network at lower prices.

Using ad hoc and epidemic routing protocol is cheap way to extend hotspot range without investing more in expensive infrastructure.

Example: Users can decide if they want to be part of ad hoc network. If they do not want to serve as relay points, they can than work only within hotspot range. If users decide to offer themselves as relay points they can use other users to ad hoc connect to hotspot (extending the hotspot range) and get cheaper access since they in a sense become part of infrastructure and are already sharing the part of the cost of operating the network by participating in it as relay point.

Ad hoc and epidemic routing in application space will enable companies to create their own small apps running on palmtops and laptops and provide addressing services, routing, billing, hot spot directory service, etc.

Other applications of ad hoc network include sensor networks with limited resources, disaster situations in which the infrastructure has been destroyed, battlefield environment or situations where setting up fixed infrastructure would be impractical as in a short duration meeting or conference.
Appendix E: System manual

Compiling and installing ns:
The development was done on a Red Hat Linux 7.3 (kernel 2.4.18-3) using gcc 2.96. (it also compiles on Red Hat 8.0, but we had several problems trying to compile it on Red Hat 9.2)
You will need at least 250Mb of disk space to uncompress, compile and install ns allinone package on your machine.

The allinone-ns tarball is available on a CD. The following commands will compile and install ns on your machine.

$ tar –xzf epi_src.ns-allinone-2.1b7a.tar.gz  unpack the ns-allinone package
$ cd  ns-allinone-2.1b7a/ change to ns directory and follow instructions from README
$ ./install  it runs configure script, make and install

The compilation may give the following error:

ns-allinone-2.1b7a/tk8.3.2/unix/tkUnixPort.h:228:20: tclInt.h: No such file or directory

To fix the error copy the tclInt.h file to the tk directory:

$ cp ns-allinone-2.1b7a./include/tclInt.h ~ /ns-allinone-2.1b7a/tk8.3.2/generic

If the install was successful, it will print out the set of instructions on how to configure the environment variables for ns. In particular, the following variables need to be set: LD_LIBRARY_PATH, TCL_LIBRARY, PATH

Group access

If several people are going to run simulations, it is possible to share a single ns installation with others. The users who are going to run ns will need to set up their environmental variables accordingly (LD_LIBRARY_PATH, TCL_LIBRARY, PATH) and have a read + executable access to the ns directory.
Routing in Partially Connected Wireless Ad Hoc Networks
Appendix F: Developer’s Manual

8.3 Classes

class toraAgent : public rtAgent

Class toraAgent is the main class containing the implementation of the actual routing protocol. It is inherited from rtAgent class and contains the following important methods and data structures:

- **copylist**: a message buffer is implemented as a directed list. Apart from messages for other nodes, buffer also keeps messages received and acknowledged by this node. This is needed to detect receiving duplicate messages and acknowledgments.
- **copylist_ack**: same for acks.
- **my_history**: history object
- **void recv()**: this procedure receives epidemic packets both from the transport level (originated from the node itself) and link level (from other nodes).
- **void prune_queue(int QD)**: prune the buffer queue according to a given queuing discipline (FIFO, Freq)
- **void fwdqadd(Packet *p)**: add a message packet to a forward queue
- **void fwdqadd_ack(Packet *p)**: add a message packet to a forward queue
- **int epihash(Packet *)**: calculate a packet hash value
- **Packet* update_copylist(Packet *p)**: add a message to a buffer, if the message already exists then update message properties, such add up allowed copies or merge hit lists
- **Packet* update_copylist_ack(Packet *p)**: same for acks
- **void msg_xchange(nsaddr_t index)**: start message exchange phase
- **void sendQRY(nsaddr_t id, Packet *p, Time delay = 0.0);**: offer a hash table (or a message) to a neighbour
- **void sendUPD(nsaddr_t id, unsigned int summ[]);**: send a list of messages we are interested in (after analysing an offer)
- **td_head nblist;**: a current list of neighbours
- **int nbcnt**: number of neighbours
- **int fwdqcnt;**: the number of messages in buffer
- **void toraAgent::rtNotifyLinkUP(nsaddr_t index)**: this is a routing API method. It is called by a link level when a new neighbour is detected
- **void toraAgent::rtNotifyLinkUP(nsaddr_t index)**: the method is called by a link level when a neighbour becomes unavailable
- **void timeout(void)**: this function is called from the timer’s callback function
- **void set_timer(double t)**: set the timer (secs)
- **void toraAgent::sendACKs(nsaddr_t id)**: send acks to a neighbour.
- **void toraAgent::rt_resolve(packet *p)**: broadcast the new packet if it’s not in the buffer

class History
Routing in Partially Connected Wireless Ad Hoc Networks

Class **History** contains methods and data structures to manipulate history information. It contains the following methods and data structures:

- `void reset` reset all info about all nodes
- `void add(nsaddr_t index, double timestamp)` add a new timestamp for a node 'index'
- `nsaddr_t getleastfreq(nsaddr_t me, toraAgent* p)` get the address of the least frequently contacted node. This information is used to purge messages a full buffer when using a history optimisation.
- `void print(void)` print history info. For debugging purposes only
- `hist_head allhistory` a history implemented as a doubly linked list. The maximal length of the list is set by a parameter `opt_history`
- `int hist_len` number of entries in history

**class TX`_timer:`**public TimerHandler

Class **TX_time** is needed to implement delays and timeouts inside simulations. When the timer expires it calls a virtual callback function `expire()` inherited from a system class `TimerHandler`.

**class TORANeighbor**

The class is keeps information about neighbour, such as node id, link_status, time of activation and other node properties. This class was not modified in XEPI.

**Files:**

Below is the short description of the files defining and implementing those classes:

- **tora.h** definition of the toraAgent, History classes
- **tora.cc** implementation of the toraAgent and History classes, `recv()` function, buffer management, garbage collection, ack generation
- **tora_api.cc** contains TORA API functions, basically a function which is called when we meet a new neighbour and a function when we loose a connection with one of the neighbours
- **tora_dest.h** implementation of tora_dest class
- **tora_dest.cc** definition of tora_dest class
- **tora_neighbour.cc** implementation tora_neighbour class
- **tora_neighbour.h** definition of tora_neighbour class
- **ns-agent.tcl** binding of tcl to c++ variables, setting up default values
- **tora_io.cc** Implementation of sendACKs method

**Compile options:**

Below is the description of all compiler options that change the logic of the protocol

- **HASH** defines whether to use hash tables during message exchange. Without hash tables, nodes will send messages one by one or offer each message separately to a neighbour (ASK)
- **BLIND** each message will keep information about all the nodes it has been routed through
- **DEBUG** print all debug information
Routing in Partially Connected Wireless Ad Hoc Networks

SHOWALL  print all the communication details
TTL      restrict the spread of the messages using time to live (the spread of the messages
can also be limited by the setting the maximum number of allowed copies a message
can have)
FWDDIRECT deliver the message only directly to the final destination. (don't give it anyone else)
ASK      affects a message exchange algorithm when we are not using hash tables. With this
parameter a node will offer a message to its neighbour and if the neighbour agrees
to accept it the node sends the message.
ACK      defines whether to use acks (XEPI). If compiled without this option we can get
original protocol.

Notes

Most of the code modified during this project was commented using an // [ACK]
comment line

Most of the original code which was modified by authors during its porting from
Monarch to ns is commented using // [EPI] comment line

Debugging ns

Debugging is absolutely necessary when trying to understand how the code works or why it doesn't
work as desired.

For debugging ns has to be compiled -g option.
It's convenient to have all simulation parameters hardcoded into the simulation script. This way one
will not have to type a long string of simulation parameters for set args each time debugging ns.

$gdb ns
=set args script.tcl
=break tora/tora.cc:156
=run

Future work

Below is the list of tasks not completed by our group, which we believe can improve the current
implementation:

- Redirect all simulation output to a trace file instead of a standard output. It should be done
  using available trace and logging functions from the trace.cc.
- Memory leaks. The source of memory leaks is in the piece of code responsible for the garbage
  collection of acknowledged messages (toraAgent::recv() method). Memory leaks can become
  a problem when running very long simulations.
- Use the same data structures for messages and acks. The reason we used a separate data
  structure for acks is to make as few changes to the original code as possible, so that it is
  easier to localize bugs. Using the same data structure for both messages and acks will make
  the code simpler and less redundant.
- Clean up the code.
Appendix G: Testing

Testing plays an extremely important role in the software development process. We used a regression and a unit testing at the end of each iteration to make sure that the program actually does what we expect it to do.

Unit tests
Each unit test involves running a simulation with a very simple scenario testing a single particular feature of the protocol such as message exchange, broadcast, counters or history. It would usually involve several nodes and one more messages travelling in the network. We implemented each unit tests as a fixed set of tcl, traffic and mobility scenario scripts for each test.

Below is the description of all unit tests that we used to check the behaviour of our protocol, together with descriptions of scenarios and the actual output of the simulation.

Scenario 1. Basic test
Nodes #0 and #1 are out of range and moving slowly towards each other. Node #1 sends a message to a node #0.
Simulation parameters: range 50

Expected behavior:
Node B should get a message from A and send an ack back when they come within range.

WARNING: using code in run.tcl to set range to 50.0
    Set xmit power to 0.00045086419753086415

Creating nodes
warning: Please use -channel as shown in tcl/ex/wireless-mitf.tcl
Agent options: buffer 30 hops 3
Loading connection pattern...
..done
Loading scenario file...
Progress (10 ticks 1000 total seconds)
Starting Simulation...
  _1_ broadcasting msg packet 0x0 to 0
H 85.100436096 _0_ contact: 1
H 85.100436096 _0_ history:
H 85.267092046 _1_ contact: 0
H 85.267092046 _1_ history:
T 85.277415882 _1_ episend 0x0 ttl 2 to 0 (broadcast)
A 85.281819977 _0_ rcvmsg 0x0 from 1 via 1
A 85.281819977 _0_ ackfwd 0x0 to 1 (broadcast)
A 85.386224071 _1_ rcvack 0x0 from 0 via 0
G 85.386224071 _1_ garbaged 0x0 buffersize = 0
A 85.386224071 _1_ buffer dump:
90 181 272 363 454 545 636 727 818 909 NS EXITING...
kew ~/epi/Epi $

Scenario 2. Epidemic test
Nodes #0, #1 and #2 are out of range of each other. Node #1 comes within a range of a node #0 and moves towards the node #2. Node A sends a message to a node #C.
Simulation parameters: range 50
Routing in Partially Connected Wireless Ad Hoc Networks

Expected behavior:

#1 should get a message from #0 and transfer it to #2. #2 sends an ack to #1.

```
num_nodes is set 3
WARNING: using code in run.tcl to set range to 50.0
    Set xmit power to 0.00045086419753086415
Creating nodes
warning: Please use -channel as shown in tcl/ex/wireless-mitf.tcl
Agent options: buffer 30 hops 3
Loading connection pattern...
..done
Loading scenario file...
Progress (10 ticks 1000 total seconds)
Starting Simulation...
   _1_ broadcasting msg packet 0x0 to 0
   90 H 171.200436096 _1_ contact: 2
   H 171.200436096 _1_ history:
   H 171.206950012 _2_ contact: 1
   H 171.206950012 _2_ history:
T 171.213865857 _1_ episend 0x0 ttl 2 to 2 (broadcast)
181 272 363 H 370.171615122 _0_ contact: 2
   H 370.171615122 _0_ history:
H 370.359744563 _2_ contact: 0
   H 370.359744563 _2_ history: 1
T 370.373227907 _2_ episend 0x0 ttl 1 to 0 (broadcast)
   A 370.37763603 _0_ rcvmsg 0x0 from 1 via 2
A 370.37763603 _0_ ackfwd 0x0 to 1 (broadcast)
A 370.48236098 _2_ rcvack 0x0 from 0 via 0
G 370.48236098 _2_ garbaged 0x0 buffersize = 0
A 370.48236098 _2_ buffer dump:
   454 545 636 727 818 909 NS EXITING...
```

Scenario 3. ack counters

Nodes 2-5 stand along a line, on a distance slightly greater than a transmission range.
Node 1 is isolated
Node 2 sends a message to node 1
Node 6 approaches node 2 and moves along line to the 5
Simulation parameters: range 50, opt_ack_refresh = 0

Expected behavior:

Messages spread along the line. Acks spread along the line without limitation

```
kew ~/epi/Epi $ ns test.tcl
num_nodes is set 11
WARNING: using code in run.tcl to set range to 60
    Set xmit power to 0.0009349119999999998
Creating nodes
warning: Please use -channel as shown in tcl/ex/wireless-mitf.tcl
Agent options: buffer 30 hops 4
Loading connection pattern...
done
Loading scenario file...
Progress (10 ticks 1000 total seconds)
```
Routing in Partially Connected Wireless Ad Hoc Networks

Starting Simulation...
_1_ broadcasting msg packet 0x0 to 0
A 0.804404133 _2_ epirecv 0x0 from 1
A 0.804404133 _0_ rcvmsg 0x0 from 1 via 1
A 0.804404133 _0_ ackfwd 0x0 to 1 (broadcast)
A 0.908808267 _1_ rcvack 0x0 from 0 via 0
G 0.908808267 _1_ garbaged 0x0 buffersize = 0
A 0.908808267 _1_ buffer dump:
A 1.100436133 _0_ ackfwd 0x0 to 1 (broadcast) counter = 0
A 1.104840267 _1_ ackdup 0x0 from 0 via 0
A 1.200436133 _1_ ackfwd 0x0 to 2 (broadcast) counter = 2
A 1.204840267 _2_ rcvack 0x0 from 0 via 1
G 1.204840267 _2_ garbaged 0x0 buffersize = 0
A 1.204840267 _2_ buffer dump:
A 1.300436133 _2_ ackfwd 0x0 to 3 (broadcast) counter = 1
A 1.304840267 _3_ rcvack 0x0 from 0 via 2
A 1.304840267 _1_ ackdup 0x0 from 0 via 2
A 1.400436133 _3_ ackfwd 0x0 to 4 (broadcast) counter = 1
A 1.404840267 _4_ rcvack 0x0 from 0 via 3
A 1.404840267 _2_ ackdup 0x0 from 0 via 3
A 1.500436133 _4_ ackfwd 0x0 to 5 (broadcast) counter = 1
A 1.504840267 _5_ rcvack 0x0 from 0 via 4
A 1.504840267 _3_ ackdup 0x0 from 0 via 4
A 1.600436133 _5_ ackfwd 0x0 to 6 (broadcast) counter = 1
A 1.604840267 _6_ rcvack 0x0 from 0 via 5
A 1.604840267 _4_ ackdup 0x0 from 0 via 5
A 1.700436133 _6_ ackfwd 0x0 to 7 (broadcast) counter = 1
A 1.704840267 _7_ rcvack 0x0 from 0 via 6
A 1.704840267 _5_ ackdup 0x0 from 0 via 6
A 1.800436133 _7_ ackfwd 0x0 to 8 (broadcast) counter = 1
A 1.804840267 _8_ rcvack 0x0 from 0 via 7
A 1.804840267 _6_ ackdup 0x0 from 0 via 7
A 1.900436133 _8_ ackfwd 0x0 to 9 (broadcast) counter = 1
A 1.904840267 _9_ rcvack 0x0 from 0 via 8
A 1.904840267 _7_ ackdup 0x0 from 0 via 8
90 181 272 363 454 545 A 629.100436136 _9_ ackfwd 0x0 to 10 (broadcast) counter = 0
A 629.104840270 _8_ ackdup 0x0 from 0 via 9
A 629.104840273 _10_ rcvack 0x0 from 0 via 9
636 A 668.967245835 _8_ ackfwd 0x0 to 10 (broadcast) counter = 0
A 668.971649969 _9_ ackdup 0x0 from 0 via 8
A 668.971649969 _7_ ackdup 0x0 from 0 via 8
A 668.971649972 _10_ ackdup 0x0 from 0 via 8
A 708.946455959 _7_ ackfwd 0x0 to 10 (broadcast) counter = 0
A 708.950860093 _8_ ackdup 0x0 from 0 via 7
A 708.950860093 _6_ ackdup 0x0 from 0 via 7
A 708.950860096 _10_ ackdup 0x0 from 0 via 7
727 A 748.811898725 _6_ ackfwd 0x0 to 10 (broadcast) counter = 0
A 748.816302858 _7_ ackdup 0x0 from 0 via 6
A 748.816302858 _5_ ackdup 0x0 from 0 via 6
A 748.816302862 _10_ ackdup 0x0 from 0 via 6
A 788.715873737 _5_ ackfwd 0x0 to 10 (broadcast) counter = 0
A 788.722982871 _6_ ackdup 0x0 from 0 via 5
A 788.722982871 _4_ ackdup 0x0 from 0 via 5
A 788.722982875 _10_ ackdup 0x0 from 0 via 5
818 A 828.554098084 _4_ ackfwd 0x0 to 10 (broadcast) counter = 0
A 828.558502218 _5_ ackdup 0x0 from 0 via 4
A 828.558502218 _3_ ackdup 0x0 from 0 via 4
A 828.558502223 _10_ ackdup 0x0 from 0 via 4
Routing in Partially Connected Wireless Ad Hoc Networks

Scenario 5. ack counter, thresh = 1
Nodes 2-5 stand along a line, on a distance slightly greater than a transmission range.
Node 1 is isolated
Node 2 sends a message to node 1
Node 6 approaches and moves along the line to node 5
Simulation parameters: range 50, opt_ack_refresh = 5, opt_ack_thresh = 1

Expected behavior: usage of counter based broadcast optimisation should reduce the number of duplicate acknowledgements.

Agent options: buffer 30 hops 4
Loading connection pattern...
done
Loading scenario file...
Progress (10 ticks 1000 total seconds)
Starting Simulation...
90 181 272 363 454 545 636 727 818 909 NS EXITING...
um_nodes is set 11
_scenario 5_ broadcasting msg packet 0x0 to 0
A 0.804404133 _2_ epirecv 0x0 from 1
Bibliography


Xiaoyan Hong, Mario Gerla, Guangyu Pei and Ching-Chuan Chiang. A Group Mobility Model for Ad Hoc Wireless Networks. Computer Science Department University of California Los Angeles, CA 90095-1596 hxy, gerla, pei, ccchiang@cs.ucla.edu

Amy L. Murphy and Xiangchuan Chen. Enabling Disconnected Communication. University of Rochester, Computer Studies Building 734, P.O. Box 270226 Rochester, New York, USA 14627, fmurphy, chenxcg@cs.rochester.edu


http://people.cs.vt.edu/%7Eirchen/microsoft-grant/Website_HTML_files/index.html