

3

# Using Location Information to Improve Routing in Mobile Ad-Hoc Networks

by

David Matthew Simmons

Submitted to the Department of Electrical Engineering and Computer  
Science

in partial fulfillment of the requirements for the degrees of  
Bachelor of Science in Computer Science and Engineering

and

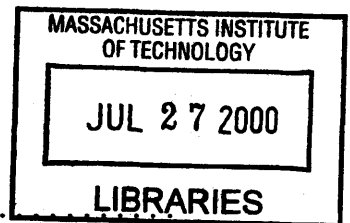
Master of Engineering in Electrical Engineering and Computer Science  
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2000

© David Matthew Simmons, MM. All rights reserved.

The author hereby grants to MIT permission to reproduce and  
distribute publicly paper and electronic copies of this thesis and to **ENG**  
grant others the right to do so.



Author .....  
Department of Electrical Engineering and Computer Science  
May 12, 2000

Certified by .....  
John J. Turkovich  
Charles Stark Draper Laboratory  
Thesis Supervisor

Certified by .....  
Robert T. Morris  
Assistant Professor of Electrical Engineering and Computer Science  
Thesis Supervisor

Accepted by .....  
Arthur C. Smith  
Chairman, Department Committee on Graduate Theses

# Using Location Information to Improve Routing in Mobile Ad-Hoc Networks

by

David Matthew Simmons

Submitted to the Department of Electrical Engineering and Computer Science  
on May 12, 2000, in partial fulfillment of the  
requirements for the degrees of  
Bachelor of Science in Computer Science and Engineering  
and  
Master of Engineering in Electrical Engineering and Computer Science

## Abstract

This thesis describes a method for utilizing position and velocity information obtained from the Global Positioning System or mission planning software to improve packet routing in a Mobile Ad-Hoc Network. The information is used to choose routes with a longer lifetime, thereby increasing the time between route discoveries and reducing routing overhead. It is also used to preemptively replace routes before they fail, providing a lower network latency to the application. Analysis of the protocol using Network Simulator shows that a reduction of almost 50% in the number of routing packets can be achieved over the performance of the Ad-Hoc On-Demand Distance Vector (AODV) protocol upon which it is based. This reduction is achieved at the expense of routing packets that are around 10% larger, and routes that average one half step longer than those chosen by AODV.

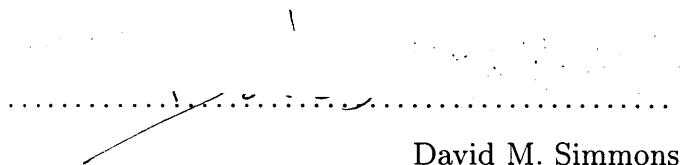
Thesis Supervisor: John J. Turkovich  
Charles Stark Draper Laboratory

Thesis Supervisor: Robert T. Morris  
Assistant Professor of Electrical Engineering and Computer Science

# Acknowledgment

This thesis was prepared at The Charles Stark Draper Laboratory, Inc., using the Draper Laboratory's independent research and development funds for Network and Communications Modeling and Simulation under contract number 15027.

Publication of this thesis does not constitute approval by Draper or the sponsoring agency of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.



.....

David M. Simmons  
May 12, 2000

[This page intentionally left blank]

## Acknowledgements

First, I would like to thank my Draper supervisor, John Turkovich, for suggesting the direction of my research and providing valuable critiques of my writing. If not for him, I probably would have left MIT without learning anything substantial about networking.

I would also like to thank my MIT advisor, Robert Morris, for agreeing to supervise this project on such short notice. As well, the technical critiques he provided allowed me to focus my thoughts and express them cleanly.

# Contents

<b>1</b>	<b>Introduction</b>	<b>10</b>
1.1	Definition of a Mobile Ad-hoc Network . . . . .	11
1.2	Differences Between MANETs and Fixed IP Networks . . . . .	11
1.2.1	Dynamic Topology . . . . .	12
1.2.2	Resources . . . . .	12
1.2.3	Security . . . . .	12
1.2.4	Unidirectional Links . . . . .	13
1.3	Applications . . . . .	13
1.4	Related Work . . . . .	14
1.4.1	Dynamic Source Routing (DSR) . . . . .	15
1.4.2	Destination Sequenced Distance Vector Routing . . . . .	16
1.4.3	Ad Hoc On-Demand Distance Vector (AODV) . . . . .	16
1.4.4	Zone Routing Protocol (ZRP) . . . . .	17
1.4.5	Location-Aided Routing (LAR) . . . . .	18
<b>2</b>	<b>Design</b>	<b>19</b>
2.1	Design Goals . . . . .	20
2.2	Communicating Position Efficiently . . . . .	21
2.2.1	Route Reply Packet Format . . . . .	22
2.2.2	Estimating Link Lifetimes . . . . .	24
2.3	Making Route Requests Smarter . . . . .	26
2.3.1	Route Request Packet Format . . . . .	27
2.3.2	Limiting Request Broadcasts . . . . .	28

2.4	Picking the Best Route . . . . .	30
2.4.1	Limiting the Final Route Length . . . . .	31
2.4.2	Receiving Multiple Route Replies . . . . .	33
2.4.3	Interactions with Expanding Ring Searches . . . . .	34
2.5	Preemptively Replacing Routes . . . . .	34
2.5.1	Accurately Determining Route Lifetimes . . . . .	35
<b>3</b>	<b>Implementation</b>	<b>37</b>
3.1	Architecture . . . . .	37
3.2	Dependencies . . . . .	38
3.2.1	Protocols . . . . .	38
3.2.2	Global Positioning System . . . . .	39
3.3	Protocol Variations . . . . .	40
<b>4</b>	<b>Results</b>	<b>42</b>
4.1	Testing Environment . . . . .	42
4.2	Packet Delivery Ratio . . . . .	43
4.3	Routing Overhead . . . . .	47
4.4	Path Lifetime . . . . .	50
4.5	Route Replacement . . . . .	52
<b>5</b>	<b>Conclusions</b>	<b>54</b>
5.1	Possible Future Work . . . . .	57
<b>A</b>	<b>Finding All Pairs Longest Lived Paths</b>	<b>59</b>
A.1	Unlimited Length Paths . . . . .	59
A.2	Limited Length Paths . . . . .	64
	<b>Bibliography</b>	<b>67</b>

# List of Figures

1-1	A hierarchical MANET . . . . .	18
2-1	Computing route lifetimes . . . . .	27
2-2	Why both hop count and lifetime are important . . . . .	32
2-3	Route request propagation . . . . .	33
3-1	Overall architecture . . . . .	38
4-1	Percentage of packets delivered at 1 m/s . . . . .	45
4-2	Percentage of packets delivered at 20 m/s . . . . .	46
4-3	Routing overhead at 1 m/s . . . . .	47
4-4	Routing overhead at 20 m/s . . . . .	48
4-5	Graph illustrating preemptive route discovery . . . . .	53



# List of Tables

2.1	AODV route reply packet . . . . .	23
2.2	New fields added to the route request and reply packets . . . . .	23
2.3	AODV route request packet . . . . .	28
3.1	Constants used in the implementation . . . . .	41
4.1	Packet delivery ratio for a varying number of connections . . . . .	46
4.2	Routing overhead for a varying number of connections . . . . .	49
4.3	Summary of the lifetime test . . . . .	50
4.4	Difference between the optimal and actual hop counts . . . . .	51

# Chapter 1

## Introduction

Computing devices have been growing smaller and more portable every year. Networking has also become more ubiquitous, from something that only large corporations would use to something that almost every computing device uses. The next logical step, networking portable devices, causes problems because most networks rely on a fixed infrastructure. However, a fixed infrastructure is generally not available in situations where a network of portable devices would be useful. A network of portable devices that does not rely on any existing infrastructure is called a Mobile Ad-hoc Network, or MANET.

One example of a MANET is a group of autonomous vehicles designed to provide reconnaissance of an area. To avoid duplication of effort and for relaying information back to a central control station, each device is equipped with a wireless transceiver. While several protocols exist that would allow these vehicles to communicate [17, 10, 2, 15], none take into account the position and velocity information that is available to such a vehicle through systems such as the Global Positioning System (GPS). In addition, some types of autonomous vehicle have a plan of their expected route. By allowing the routing and path finding programs to communicate, several benefits can be realized. These benefits include lower latency in packet delivery or path finding that tries to keep network links active. This thesis explains the current state of research on MANETs and proposes additions to the Ad-hoc On-Demand Distance Vector protocol[17, 14] to allow it to utilize position and velocity

information when discovering routes. The information is used to find routes that remain active longer and to preemptively replace failing routes. This new protocol's design is described in chapter 2. Chapter 3 presents a sample implementation with an emphasis on portability from a simulation environment to a production system. In chapter 4, several variations of the protocol are analyzed. Finally, chapter 5 provides thoughts on when these additions would be useful and what benefits they provide.

## **1.1 Definition of a Mobile Ad-hoc Network**

A mobile, ad-hoc network (MANET) consists of a group of devices with wireless transceivers in an environment without a preexisting network infrastructure. The devices, or nodes, may be arranged in such a way that the diameter of the network is greater than the transmission range of any one node. In addition, the nodes may roam, moving in and out of transmission range of each other in the network. A good MANET routing protocol will be able to discover and maintain multihop routes through the network with minimal information about the network.

## **1.2 Differences Between MANETs and Fixed IP Networks**

Packet routing in non-mobile Internet Protocol (IP) based networks such as the Internet has been well studied for decades. Although it may seem similar to MANET routing at first, in reality, it is a completely different problem. While there are several protocols that attempt to integrate a mobile IP node into the existing Internet [16, 19], they are not suited for a completely infrastructure-less environment. This is because they rely on being able to forward packets to the mobile node from a node with a known fixed location on the network. When no nodes have a fixed location, this scheme fails. In addition, there are other design considerations that standard IP routing schemes do not address. These include dynamic topology, resource management, and security. A brief description of these problems follows.

### 1.2.1 Dynamic Topology

Because the nodes are mobile, the topology of the network is continually changing. This causes links between nodes to be created and destroyed, which makes it impossible to establish permanent routes between any two nodes. At first, this seems similar to what happens in an IP network when a link goes down, and the network routes around the problem. However, in the MANET case, this happens with much greater frequency than in a fixed network. IP networks also depend upon a hierarchical naming scheme (IP addresses) where a name is related to the location of the node in the network. This is impossible to enforce sensibly in a MANET, where the nodes are allowed to move freely.

### 1.2.2 Resources

Another consideration in designing a MANET is that nodes typically have limited resources. First, they are generally running on battery power, so the longer a node has to transmit, or even listen, the faster the battery is depleted. Therefore, to extend battery life, the less the node participates in the network, the better. Second, some wireless devices have only a small amount of memory and computation power. In these devices, it is undesirable to store large routing tables, so a minimal amount of information should be in the table. Similarly, with a limited amount of computing power, you need to ensure that the networking code needs to leave enough free cycles for the device to do other computations. (Unless, of course, the device doesn't need to do anything besides route packets.) Third, the wireless medium is constrained, usually having no more than, and frequently less than, 10 Mbps of bandwidth. The routing protocol must not take up a significant portion of the already limited bandwidth.

### 1.2.3 Security

In a wireless network, any intruder with appropriate equipment can intercept all communications, unlike in a wire based network, where they must have physical access to the network links. This makes encryption a requirement to prevent eavesdropping.

Security can also be compromised by a node using malicious routing code. By sending out false routing updates, it is possible to create denial of service attacks. This problem would be solved with an authentication scheme where only trusted nodes would be allowed to participate in the network and routing messages from all other nodes would be ignored. Unfortunately, requiring authentication would be undesirable in applications where the participants are unknown before the network is deployed. The issue of security in a MANET is extremely broad and is not discussed in this thesis. See [22] for issues relating to MANET security.

### 1.2.4 Unidirectional Links

MANETs can also differ from standard networks by having unidirectional links. Internet routing protocols usually assume that if data can be transmitted in one direction between two nodes, it can also be transmitted in the other direction. This is not always true in a wireless network, where environmental conditions could cause a link to be unidirectional. However, the IEEE 802.11 standard<sup>1</sup> requires links to be bidirectional for its medium access control (MAC) protocol distributed coordination function (DCF) to work. Before any unicast packet is sent, a Request-to-Send / Clear-to-Send (RTS / CTS) exchange takes place between the nodes at either end of the link. The source node will not send the packet until it receives the CTS packet from the destination. Because of this, many MANET protocols, including the basis for this thesis, the Ad-Hoc On-Demand Distance Vector protocol (AODV), only use bidirectional links.

## 1.3 Applications

A MANET's dynamic nature and its lack of fixed infrastructure make it ideal for situations when a network needs to be set up quickly and easily. Some of these situations are:

---

<sup>1</sup>A common standard specifying the requirements for the physical and medium access control layers of a wireless transceiver. See [5].

**Conferences** If the attendees wished to share data, and the conference environment did not have network wiring already installed, a MANET could be used to distribute information quickly.

**Tactical Military** The military depends on communication between its units in the field and its command and control centers. MANETs allow individual units to stay connected, even while conducting operations.

**Sensor Networks** Groups of intelligent sensors which need to communicate would also benefit from a MANET. If one sensor becomes disabled, the network would be able to easily work around the problem, rather than partitioning the network.

## 1.4 Related Work

The Internet Engineering Task Force (IETF) currently has a working group discussing MANETs<sup>2</sup> to investigate possible MANET routing specifications. Their goal is to study and refine the current protocols. Eventually, finalized versions of the protocols will be published as standards so that many different implementations of the same protocol can communicate seamlessly. Currently, no MANET routing protocols have been completely standardized. However, the working group has published a Request for Comments (RFC)[7] that defines the relevant characteristics of a MANET and identifies metrics useful for measuring the performance of a routing protocol. As noted in the RFC, one protocol may not be optimal for all situations. In a real life situation, a MANET may only have a subset of the constraints listed above. For example, not all wireless devices are constrained by computing power and memory. In this case, it would be possible to choose a less memory efficient and more processor intensive algorithm that provides better network performance.

Included below are summaries of some of the representative MANET routing protocols. What follows is only a brief description, and should not be used when

---

<sup>2</sup>see <http://www.ietf.org/html.charters/manet-charter.html>

implementing the protocol. The references given in each section provide a much more detailed and correct description of each algorithm.

### 1.4.1 Dynamic Source Routing (DSR)

Dynamic Source Routing[2, 11] is a protocol that specifies the entire route of a packet in its headers. Each node maintains a complete route to each of the nodes it wishes to route to in its routing table. The table is maintained via two protocols, route maintenance and route discovery. The route maintenance protocol is used to detect broken routes, while the route discovery method is used to find new routes and fix broken ones.

A route discovery begins when a node that needs a route to a remote host sends out a *route request* packet to all of the nodes within transmission range. The route request contains the requesting node's address, the address that it wishes to route to, and a unique identifier. As this packet propagates through the network, it builds up a list of the nodes it has traversed. When a node receives a route request packet, it first checks to see if it has seen the packet's unique identifier recently. If so, or if this node's address is already listed in the packet, the packet is silently dropped. Otherwise, if the host's address matches the destination in the route request, it sends the route listed in the route request back to the originating node via a *route reply* packet. The answering host obtains this route back to the requester either from its route cache, or by reversing the route in the route request packet. For all other route request packets, the node simply appends its address to the end of the route record and transmits the packet to all of its neighbors.

Detecting when a link goes down can be a difficult problem in a MANET. Depending on the hardware in use, the link layer may be able to detect broken links using hop by hop acknowledgments. If this is not available, the software can detect link breakage by using *passive acknowledgement*. Because most packets must be retransmitted at each hop, a node can determine if its link still works by listening for the next hop node to retransmit the packet.

If a break is detected, the host that detected it sends a *route error* packet to the

host trying to route through that link. This packet contains the addresses of the two hosts at either end of the link. When a host receives a route error packet, it removes that link from its cache, and begins a new route discovery process to find a way around the problem.

### 1.4.2 Destination Sequenced Distance Vector Routing

Unlike DSR, Destination Sequenced Distance Vector (DSDV) routing[15] does not use source routes in its packets. Each node in a network running DSDV maintains a table of the possible destinations, and the next hop node to reach that destination. Each entry also contains the number of hops to the destination and a sequence number, which is used to ensure that the freshest route is always used.

Route discovery is proactive, rather than reactive, as in DSR. Each node periodically broadcasts to each of its neighbors its complete routing table. Route changes are also broadcasted when a node detects a topology change. Every time a node broadcasts its routing table, it attaches a higher sequence number. If a node ever receives a routing update with a lower sequence number than it had previously seen for that destination, it discards the routing update.

### 1.4.3 Ad Hoc On-Demand Distance Vector (AODV)

Ad Hoc On-Demand Distance Vector routing[17, 14] is much like DSDV in that each node only knows the next hop to a destination. However, unlike DSDV, AODV route discovery is reactive: routes are only discovered when they are needed by some upper level application. This eliminates the need to use network bandwidth to broadcast routing tables, as in DSDV. AODV was also designed to be used with IP addressing, which has several benefits, including seamless operation with TCP (for a discussion of TCP/IP, see [20]). AODV makes routing decisions based on two new user datagram protocol (UDP) packets it generates, the *route request* packet (RREQ), and the *route reply* packet (RREP).

Whenever an IP packet is seen with a destination that is not in the route cache,



the host generates a route request packet. This packet contains the IP address of the originating node, the IP address of the destination node, and unique id, much like the route request packet in DSR. Like DSR, the packet is broadcasted to all the nodes within transmission range. When a node receives a RREQ, it first checks the unique id to make sure that it hasn't seen it before. Any duplicate route request packets are dropped. The receiving node then consults its routing table. If it finds a working route to that destination, it sends a route reply back to the originating node. If it doesn't have a route to that destination, it decreases the time to live (TTL) field in the IP header, and broadcasts the RREQ to all of its neighbors. In either case (replying to, or forwarding the RREQ), the node adds a reverse route to the source of RREQ to its routing table, using the previous hop of the RREQ as the next hop. This route allows the node to send any reply it receives back to the source node.

The route reply packet is then propagated back to the original host. At each step, the node adds the next hop node to its route cache, and then transmits the packet backwards again, listing itself as the next hop. Unlike DSR, which lists the entire route, only the next hop node is listed in the route reply packet. As in DSDV, an increasing sequence number is given to each route to prevent stale routes from overwriting newer ones.

#### 1.4.4 Zone Routing Protocol (ZRP)

None of protocols discussed above try to impose a hierarchy on the nodes in the network, unlike IP. For large networks, some type of hierarchy is needed to allow the network to grow while still having adequate performance. It is possible to organize the nodes into clusters, each having a node designated as a *cluster head* (See figure 1-1.). The cluster head is responsible for routing messages from its cluster to nodes in other clusters. To do this, the cluster heads are grouped into a second level network. To route a message from node one in cluster A (A-1) to node three in cluster B (B-3), node A-1 first decides that node B-3 is not in its cluster, so it forwards to cluster head A-0. In the network of cluster heads, A-0 then finds a route to the cluster head for cluster B, B-0. Once B-0 receives the packet, it can route it to node B-3, which

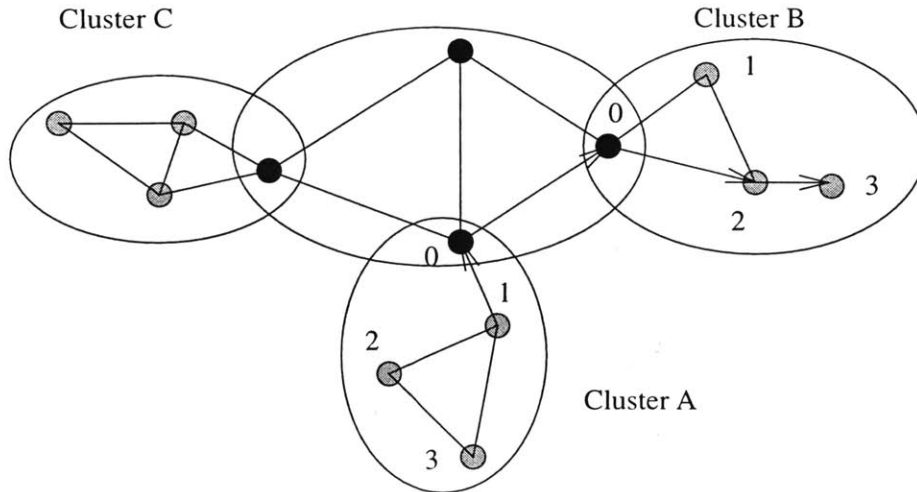


Figure 1-1: An example of a hierarchical MANET. The darker nodes are cluster heads, used for routing packets between clusters.

is in its cluster.

This scheme is problematic because it imposes a rigid hierarchy on the network, rather than a flexible one. Nodes cannot migrate from cluster to cluster easily. If a cluster head goes down, replacing it can be expensive in terms of network traffic, because every node in the cluster must agree on the new cluster head. Also, the cluster head may have specialized hardware that allows it to communicate at a greater distance. The Zone Routing Protocol[10] attempts to solve these problems by imposing a different hierarchy on the MANET. Each node is implicitly in a cluster of a fixed number of hops around it. In each cluster, routing tables are distributed in a proactive manner, much like DSDV. Routing from cluster to cluster, however, utilizes a reactive approach to minimize the size of the routing tables that are broadcasted.

### 1.4.5 Location-Aided Routing (LAR)

Location-Aided Routing (LAR)[12] uses location information to limit the search for a node. Once a node is found, its position is noted. If communication with the node is lost, the new search for the node is limited to the area that the node previously occupied. This helps conserve bandwidth, especially as the network grows to a large number of nodes.

# Chapter 2

## Design

In some MANETs, the nodes might have access to their current speed and location, either using inertial navigators, Global Positioning System (GPS) transceivers, or by other means. This information can be used to help predict how long two nodes will be in contact. While just velocity and location is useful in some situations, such as when the nodes change direction infrequently, more detailed information is needed in other situations. In a system of military vehicles, each node might have a detailed knowledge of its future path from its mission planning software. This information can be used to provide a much better estimate of network connectivity than simply the current position and velocity.

Once an estimate of the network connectivity is obtained, it can be used in several different ways. First, in the route discovery phase, the lifetime of each route can be estimated. This allows longer lived routes to be favored over shorter lived ones. Secondly, once a route's lifetime is known, it can be pre-emptively replaced with another route before it fails. Lastly, if an approximate position of the destination is known, the range of a route request broadcast can be limited geographically, saving bandwidth in other parts of the network. The last application was researched in [12]. This thesis focuses exclusively on the first two applications.

If the network can pick routes that stay active significantly longer, this confers several benefits. First, the route discovery process is expensive in terms of both time and bandwidth. Whenever a route breaks, it can take up to a full round trip time or

more between source and destination before a new route is activated. Since finding a new route involves broadcasting to the entire network, it also wastes bandwidth. Even a small reduction in the number of full network broadcasts is welcome. Second, reducing the amount of routing packets on the network means that more bandwidth is available for applications. In the limited bandwidth environment of current wireless technology, any bandwidth savings is useful.

Pre-emptive route discovery is useful for certain applications, such as live data feeds. If a route breaks, communication with the source of the data is lost while the route is being re-established. This might result in lost data. By replacing routes before they fail, two nodes can be in continuous contact.

## 2.1 Design Goals

In designing the routing protocol, the constraints of a MANET must be considered. Therefore, the protocol tries to achieve the following goals.

**Small Routing Tables** The routing table should consume as little memory as possible to allow it to run in as many types of devices as possible.

**Low Computation Requirements** Similarly, each node should only have to devote a small number of cycles to the routing protocol. As many cycles as possible should be available to running other processes.

**Low Routing Overhead** The purpose of a MANET is route application packets. The routing protocol should consume as little bandwidth as possible to leave more bandwidth for applications.

The remainder of this chapter describes a method for adding position and velocity information to AODV (see §1.4.3 and [17, 14]) while trying to meet these goals. AODV is a good starting point for meeting these goals for several reasons. First, because it is routed hop by hop, the routing table at each node is small compared to a routing table for source routing. This is because for each route in the table, only

the next hop address and some housekeeping variables need to be stored. In contrast, a source routed protocol will need to store the entire path as well as the housekeeping variables. Similarly, source routed packets are larger than packets routed hop by hop, and therefore consume more energy to transmit and receive. Because this algorithm might choose routes with longer hop counts, the energy savings provided by hop by hop routing is needed. Hop by hop routing also gives the ability to locally repair breaks in a route, which can reduce the amount of routing overhead. The distributed nature of hop by hop routing also provides a good foundation for spreading the computation required to compute a route's lifetime across several nodes. The only significant disadvantage to choosing a hop by hop protocol is that in a source routed protocol, each node has greater control over the routes it chooses. This makes avoiding routing loops much simpler, and allows the source node to choose its own routes, as opposed to the route being decided upon by a committee of nodes.

## 2.2 Communicating Position Efficiently

The first feature that is needed before the lifetime of a route can be calculated is a way to communicate the position and velocity of each node. One way to do this in a MANET is to have each node broadcast its position and velocity at a fixed interval or whenever its velocity changed. Each node can then build a graph in memory that represents the current physical state of the network. Using this graph, each node can source route each packet using a path that it believes to be valid.

This method has several drawbacks. First, although both the network state and a hop by hop routing table can be stored  $O(n)$  ( $n$  nodes in the network) space, computing the best routes given the network state is expensive. Solving for either the best route between two nodes or the best routes between all pairs of nodes both require  $O(n^3)$  time (see appendix A). With a routing table, figuring out the best routes sourced from the current node to every other node in the network requires  $O(n)$  time. In most cases, only one of those routes is needed, reducing the amount of computation needed for a routing table lookup to  $O(1)$  time. The same lookup

with only network state information would require  $O(n^3)$  time. The results of this computation could be cached, saving computation time, but then the memory usage would increase.

Second, depending on the network dynamics, broadcasting each velocity change could use a significant amount of the network's bandwidth. By only requesting this information in an on demand manner, the bandwidth used is reduced as long as the average time between each route request is greater than the average time between velocity changes.

This thesis examines embedding navigation, position, and velocity information into the preexisting AODV routing packets. This technique gives the advantages of quick routing table lookups and possibly reduced bandwidth over broadcasting each velocity change. When a node sends a routing packet, it includes its current position and velocity. The upstream node can then make an estimate of how long the link will remain in existence. A new field indicating the projected lifetime of the entire route is also added. At each hop, the receiving node updates this field to be the minimum of its current value and the estimated lifetime of the link it just traversed. The details of these changes are given in the next two sections.

### 2.2.1 Route Reply Packet Format

Table 2.1 shows the original AODV route reply format[17]. The type field is a constant identifying the packet as a reply packet. The destination IP address and sequence number give information about the destination of this route. The source IP address is the address of the node requesting the route. The lifetime field is a guess at the lifetime of the route based on constants in the implementation. Each new route is given a default lifetime when it is created. As it ages, if it is not used, that lifetime is decreased.

Extending this packet format to allow it to carry location and velocity information requires adding six fields, as shown in table 2.2. These fields are dynamically updated by each node that the packet passes through. The first three fields are the node's coordinates in some agreed upon space. The next three fields give the node's current

Type (7 bits)	Reserved (17 bits)	Hop Count (8 bits)
Destination IP Address (32 bits)		
Destination Sequence Number (32 bits)		
Source IP Address (32 bits)		
Lifetime (32 bits)		

Table 2.1: Table showing the format of an AODV route reply packet

X Position	Y Position
Z Position	$dx$ Speed
$dy$ Speed	$dz$ Speed

Table 2.2: Table showing the new fields added to both the route request and reply packets. All fields are between 16 and 32 bits wide.

velocity, in multiples of the unit vectors. These fields are as wide as needed to convey sufficient precision to allow accurate predictions. Between 16 and 32 bits should be enough for most applications. In addition to adding these fields, the semantics of the original lifetime field is changed. Whereas previously it remained static as the route reply traveled back to the source, now it must be updated at each hop in the new protocol. When the route reply is first generated, the lifetime field is set to infinity if the node creating the packet is the destination of the requested route. Otherwise, the replying node must be replying using an entry in its routing table. In this case, it sets the lifetime field to the estimated lifetime of the route in its table. Then, as the route reply is sent back to the requesting node, two important things happen at each node along the way. First, the lifetime field is set to the minimum of its current value and an estimate of the lifetime of the link the reply just traversed. This calculation is described in §2.2.2. Calculating the lifetime of a route in this fashion distributes the computational burden across the entire route, rather than concentrating it at any one node. Secondly, the six new fields are updated by querying the part of the system responsible for keeping track of position and velocity. After these two updates are performed, the packet is sent to the next hop closer to the requesting node.

Mission planning information can be added in a similar fashion. In this implemen-

tation, it is simply added as a list of waypoints and the velocity to the next waypoint. Although this method is extremely inefficient, in a real system that utilizes mission planning information, the information could be compressed considerably using methods discussed in the last paragraph of §4.3. When a node receives a packet containing mission planning information, it compares its plan with the plan of the downstream node using the method given in the next section. The lifetime field is then updated as it was with just location and velocity information. The benefit of using mission planning information is that more accurate predictions of how long two nodes will be in contact can be obtained.

### 2.2.2 Estimating Link Lifetimes

An easy way to calculate the lifetime of a link is to assume that the radio has a certain range, and then use the distance formula to determine when the distance between the two nodes will be greater than that range. Assume that there are two nodes,  $n_1$  and  $n_2$ , one at  $(x_1, y_1)$  and the other at  $(x_2, y_2)$ , moving at velocities  $(dx_1, dy_1)$  and  $(dx_2, dy_2)$ <sup>1</sup>. The distance between them in terms of  $t$ , the time elapsed, is given by:

$$d(t) = \sqrt{((x_1 + t \cdot dx_1) - (x_2 + t \cdot dx_2))^2 + ((y_1 + t \cdot dy_1) - (y_2 + t \cdot dy_2))^2} \quad (2.1)$$

To solve for the value of  $t$  when the nodes are separated by a distance of  $d_0$ , we set  $d(t) = d_0$ , and rewrite in terms of  $t$  (see equation 2.2). Once it is in this form, it can be easily solved using the quadratic formula.

$$\begin{aligned} d_0^2 &= ((x_1 - x_2) + t(dx_1 - dx_2))^2 + ((y_1 - y_2) + t(dy_1 - dy_2))^2 \\ &= (x_1 - x_2)^2 + 2t(x_1 - x_2)(dx_1 - dx_2) + t^2(dx_1 - dx_2)^2 \\ &\quad + (y_1 - y_2)^2 + 2t(y_1 - y_2)(dy_1 - dy_2) + t^2(dy_1 - dy_2)^2 \end{aligned}$$

---

<sup>1</sup>The third dimension was omitted to limit the equations to a reasonable length. It is easily added in an actual implementation.



$$0 = t^2((dx_1 - dx_2)^2 + (dy_1 - dy_2)^2) + (x_1 - x_2)^2 + (y_1 - y_2)^2 - d_0^2 \quad (2.2)$$

$$+ 2t((x_1 - x_2)(dx_1 - dx_2) + (y_1 - y_2)(dy_1 - dy_2))$$

When a node receives a route reply packet, it calculates the lifetime of the link the reply just traveled using the information in the packet, as well as its own information. After solving equation 2.2, it then compares it with the lifetime field of the packet. If the computed value is less than the value in the packet, it overwrites the packet's value. Then, it overwrites all the fields in table 2.2 with its own information, and forwards the reply to the next hop.

Calculating the length of time that two nodes will stay in contact when given mission planning information is more complicated. Let a route be represented by a list of 8-tuples of the form  $(\alpha, \omega, x, y, z, dx, dy, dz)$ . Each such tuple represents a segment of the route where the node is moving in a straight line with constant velocity. The interval in time,  $[\alpha, \omega]$ , represents the time that this tuple describes the movement of the node. At the time  $\alpha$ , the node is located at  $(x, y, z)$ . For brevity, let expressions of the form  $\alpha_{nm}$  mean "the  $\alpha$  term of the  $n^{\text{th}}$  tuple (indexed from zero) of route  $m$ ". The  $m$  subscript will be omitted when there is no ambiguity as to which route is being discussed.

For any route  $m$ , several invariants are always true:

$$\forall n < \text{length}(m), \alpha_n < \omega_n \quad (2.3)$$

$$\forall n < \text{length}(m) - 1, \omega_n = \alpha_{n+1} \quad (2.4)$$

$$\alpha_0 = 0 \quad (2.5)$$

$$\omega_{\text{length}(m)-1} = \infty \quad (2.6)$$

$$\forall n < \text{length}(m) - 1, x_{n+1} = x_n + (\omega_n - \alpha_n)dx_n \quad (2.7)$$

Invariant 2.3 ensures that each tuple describes a positive, non-zero amount of time. The next invariant, 2.4, states that there are no gaps in the timeline of the route. The next two indicate that all routes start at time zero, and continue to time

$\infty$  (or the end of the simulation). The last invariant, 2.7 (which also holds for  $y$  and  $z$ ), states that each new waypoint picks up exactly where the previous one left off. Nodes are not allowed to “jump” from one spot to another instantaneously. So, at a given time  $t$ , there exists an  $n$  such that  $\alpha_n \leq t \leq \omega_n$ . At this time, the position of the node is  $(x_n + t \cdot dx_n, y_n + t \cdot dy_n, z_n + t \cdot dz_n)$ . These facts can be used to write a function,  $\psi(m, t)$ , that takes a route description,  $m$ , and returns the node’s position and velocity at time  $t$ .

Using the above machinery and equation 2.2, the time that two routes,  $a$  and  $b$ , will be within transmission range can be calculated. Let  $\phi(\psi(a, t), \psi(b, t))$  be the result of solving equation 2.2 for  $t$ , given the position and velocity of  $a$  and  $b$  at time  $t$ .  $\phi$  returns the amount of time that  $a$  and  $b$  are in contact when they start from their locations at time  $t$ , and travel with constant velocity. Using  $\phi$  and  $\psi$ , the pseudo code in figure 2-1 returns the amount of time that  $a$  and  $b$  will be in contact. In the code, the notation  $a[i] \rightarrow \omega$  is equivalent to  $\omega_{ia}$ . The code works by breaking time into intervals where neither  $a$  nor  $b$  change velocity. It then applies  $\phi$  to the start of that interval. If  $\phi$  returns a value that indicates that the nodes will be in contact for the entire interval (line 11), it adds the the length of time of this interval to the running total and advances to the next interval. Once  $\phi$  indicates that the nodes will go out of contact in the current interval, this value is added to the running total and returned (line 22).

## 2.3 Making Route Requests Smarter

Whenever a node needs a new route, it discovers it via the route request and route reply packets. However, nodes also add routes to their routing tables when they receive route *request* packets, so that they can route the reply properly. These *reverse routes* are usually expired quickly. Using the same techniques that were proposed in §2.2, a better estimate of how quickly these routes should be expired can be obtained. Instead of appending path information onto just route replies, it can also be appended to route request packets. By extending the expiration time of reverse routes, more

---

```

double
link_time(move *a, move *b) {
    double t;
    int i = 0;      /* the current index into a */
    int j = 0;      /* the current index into b */
    double ctime = 0; /* the start of the current interval */
    double ans = 0; /* our running total of the answer */

    while (finite(a[i]-> $\omega$ ) || finite(b[j]-> $\omega$ )) {
        t =  $\phi(\psi(a, ctime), \psi(b, ctime))$ ;
        if ((t + ctime) > MIN(a[i]-> $\omega$ , b[j]-> $\omega$ )) {
            if (a[i]-> $\omega$  < b[j]-> $\omega$ ) {
                ans += a[i]-> $\omega$  - ctime;
                i++;
                ctime = a[i]-> $\alpha$ ;
            } else {
                ans += b[j]-> $\omega$  - ctime;
                j++;
                ctime = b[j]-> $\alpha$ ;
            }
        } else {
            ans += t;
            return ans;
        }
    }
    return ans +  $\phi(\psi(a, ctime), \psi(b, ctime))$ ;
}

```

---

Figure 2-1: Computing route lifetimes

routes will be in each node's routing table at any given time. This implies that when a node needs a route, there is a higher probability that it is already in the routing table. Adding estimated life information to route request packets also allows requests to travel exclusively over long lived links, resulting in longer lived routes (see §2.3.2).

### 2.3.1 Route Request Packet Format

The changes made to the route request packet are similar to the modifications made to the route reply packet. The original route request packet is shown in table 2.3. The type and hop count fields have the same purpose as in the route reply packet. The broadcast ID field is unique to each request from a given source and is used to

Type (7 bits)	Reserved (17 bits)	Hop Count (8 bits)
Broadcast ID (32 bits)		
Destination IP Address (32 bits)		
Destination Sequence Number (32 bits)		
Source IP Address (32 bits)		
Source Sequence Number (32 bits)		

Table 2.3: Table showing the format of an AODV route request packet

determine which requests a node has already processed. Every time a node receives a route request, it remembers the broadcast ID. If it sees the same ID again, it drops the packet. The destination IP address and destination sequence number are criteria for the requested route. The source of the request is looking for a route to the destination IP address with a sequence number greater than the one in the request. The sequence number prevents nodes from replying with stale routes and possibly generating routing loops. The source IP address and sequence number are used to set up a reverse route back to the source of the request (see §2.3).

All of the fields shown in table 2.2 are added to the route request as well. In addition, two other fields are added. An `estlife` field contains the estimate of the lifetime of the route this request has traversed, much like the lifetime field of the route reply. The other new field is a `minlife` field. This is used to ensure that the request results in a reply that has a given minimum lifetime. At any point, if the estimated lifetime field of the request drops below the minimum lifetime field, the request is immediately dropped. If a request is dropped in this manner, its broadcast identifier is not remembered, so that the node can process another request with the same broadcast identifier and a longer lifetime. The rationale behind this decision is given in the next section.

### 2.3.2 Limiting Request Broadcasts

AODV is built on the assumption that the replies returned in response to a route request will have low hop counts. Every time a route reply is generated the route

request stops propagating, which ensures that each route reply returned has a different route prefix. For each route request, it is rare to receive more than two replies. Often, these few replies do not contain routes substantially better than the route that AODV would pick. One way of increasing the number of replies would be to allow each node to generate multiple replies, as long as each reply was for a better route than the previous one. This will generate some extra replies, but is not an ideal solution because a particular node can only generate multiple replies for routes that use a completely distinct set of nodes (with the exception of the querying and replying node). This requirement causes many viable routes to be disregarded. To work around this problem, each node would have to forward requests that it perceived to be better than any previous request seen with the same broadcast identifier. Since each subsequent request tends to have a longer lifetime, this causes an explosion in the amount of bandwidth used.

The `minlife` field in the route request packets allows us to get around this problem. When the requesting node sets this field, each receiving node first calculates the estimated lifetime of the previous link using the techniques described in §2.2.2. If this calculation returns a lower number than the one in the minimum lifetime field, the packet is dropped. Because of this, route requests only propagate over links that have a lifetime at least equal to the minimum lifetime field of the request. Because the route returned from the route request is the path that the request takes, this ensures that the route formed has a lifetime of at least the minimum life specified in the request. In essence, the minimum life field takes the graph of the network and removes all the links with low lifetimes. Then the AODV algorithm is applied to the new graph.

However, there are some caveats with this technique. If no route can be found with the minimum life, the route request process will fail, and have to be restarted. This can be avoided by carefully setting the minimum life field is set to a reasonable value (15 seconds worked well for most of the test cases). If the route request fails anyway, it can be easily fixed by sending out another request without the minimum life field set, thereby mimicking the behavior of AODV. The other danger is that the

route returned will have an excessive hop count. This problem is only countered by setting the minimum life field as low as possible, while still having it high enough to discard extremely short lived routes. In general, the setting for the minimum life field should be carefully tweaked for a particular network configuration.

The other problem stems from the fact that AODV stores each route in a distributed fashion. If two nodes (A and B) query for the same destination with a different setting for the minimum life field, in the worst case, only one will get a route with the desired characteristics. If B queries second, B's querying process might overwrite the routing entries set up by A. From that point on, any packet sent by A would travel along the route set up by A until that route intersects with B's route. The packet would then be routed to the destination along the remainder of B's route. However, this will never leave the routing tables in an inconsistent state. In the scenarios discussed in chapter 4, this problem never occurs because all the nodes search with the same settings for the minimum life field, as they would in a tightly integrated system.

This implementation only sets the minimum life field on the first request for any route. If a second request needs to be sent, the field is set to zero, which returns the same route that AODV would return.

## 2.4 Picking the Best Route

Even if the minimum life field is set, multiple replies might be returned. Picking the best of those returned is important. The easiest thing to do would be to pick a route just based on how long it is predicted to be alive. However, this method might pick extremely long routes, which could possibly waste excessive amounts of bandwidth. The solution is to take into consideration both the hop count and the predicted lifetime of a route into account. To determine if one route is better than another, we first check to see if the new route's hop count is within a given tolerance of the previous route's hop count. If it is, then the route with a longer lifetime is

better. If not, then the one with a shorter hop count is better<sup>2</sup>.

The hop count tolerance value must be chosen very carefully. At its most conservative setting (zero), no extra bandwidth will be used by choosing paths that are longer than usual as long as the minimum life field is not set. However, once the minimum life field is increased beyond zero, the possibly extra hops in the routes chosen could end up using more bandwidth than the savings provided by not having to re-initiate the route discovery process. In the case where the minimum life field has been set to a value greater than zero, the tolerance should be very low. This is because once the minimum life field is set, it is possible that a route with the shortest number of hops between the source and destination will not be seen. Even choosing a tolerance of zero in this case might still result in routes with more hops than the best that AODV would pick.

Figure 2-2 shows why picking a route based entirely on lifetime is not always desirable. In this example, the shaded nodes are stationary, while the unshaded ones are mobile. The node labelled “src” wishes to communicate with the node labelled “dest”. There are two choices. First, it can set up a route through all the stationary nodes. This route will be predicted to stay up forever, because none of the nodes involved are moving. The second choice is establishing a route through node ‘A’. This route will be much shorter lived, but it will also have a much lower hop count. However, there is a good chance that by the time it fails, a new route will be available through node ‘B’. Although this is an extreme example, it shows the importance of not relying completely on lifetime of a link, but also the hop count.

### 2.4.1 Limiting the Final Route Length

In the current scheme, every time a reply is heard, the route in the reply is compared to the entry in the routing table. If the reply’s hop count is within a tolerance of the route in the table, and its lifetime is longer, the entry in the table is replaced. As this process repeats, the entry in the routing table could eventually have a very

---

<sup>2</sup>This is not entirely correct. See §2.4.1 for a discussion of the problem.

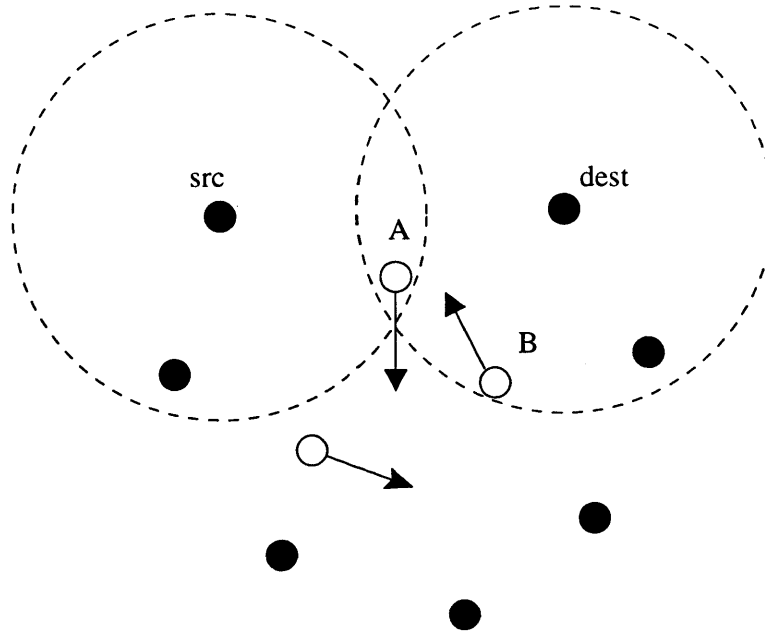


Figure 2-2: Example illustrating why the hop count must be taken into account when choosing routes based on their lifetime. The dark nodes are stationary and the light one are moving in the direction indicated.

large hop count. This becomes even more of a problem if the minimum life field is set too high, forcing only routes with long hop counts to be picked. To fix this problem, another field is added to the routing table to keep track of the shortest distance to the destination seen so far. Now, when a route reply is received, its hop count is compared with the best hop count that has been seen, instead of the hop count of the current route. This best hop count is updated at every route reply. If a reply is received that indicates a shorter route to the destination is possible, the best hop count field is overwritten even if the route in the reply does not replace the current route.

This solution could also cause problems. Suppose the routing daemon is programmed to accept routes with a hop count of only one greater than the best hop count. After it sends out a route request, it receives three replies. The first lists a hop count of three, and a lifetime of one minute. The second lists a hop count of four, but a lifetime of two minutes. Because the second's hop count is within the allowed tolerance of hop counts and its lifetime is greater, it replaces the first route in the routing



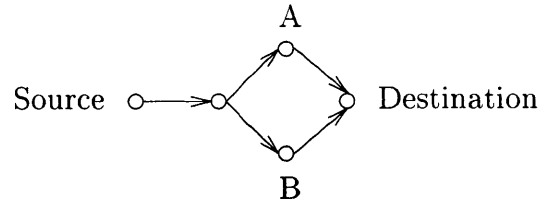


Figure 2-3: Diagram showing the propagation of a route request from source to destination. The destination will drop either the packet from A or B, and send exactly one reply back to the source.

table. The best hop field remains set to three. A third reply is received which lists a hop count two, and a lifetime of ten seconds. Between these three routes, the first is the most desirable according to our heuristic. However, because of the order they were received, the third route will be used. This could be solved by remembering several possible routes for each destination. However, it is a reasonable assumption that replies with lower hop counts will reach the requesting node first, making situations such as the above unlikely. Therefore, it is safe to ignore this problem.

### 2.4.2 Receiving Multiple Route Replies

Even after specifying a minimum acceptable life for the returned route, it is advantageous to receive as many route replies as possible, so that the one with the best hop count and lifetime combination can be chosen. This can only happen if many nodes have routes to the destination in their routing tables. Figure 2-3 shows what happens when a node tries to discover a route. All routing tables are initially empty. The source node broadcasts a route request packet. The destination node receives two copies of this packet: one from node A, and the other from node B. The request arrives from node A at the destination first. The destination will reply to the request, sending a route reply packet back to the source via node A. It will then receive the request from B, which it will discard, because it has already seen another copy of the request.

The route from source to destination via B might be longer lived, but it will not be chosen because the source will not receive a route reply along that route. One possible remedy for the problem is to have B reply to the route request, rather than forwarding

it to the destination. However, B will only reply if it has a route to the destination in its routing table. In general, if more nodes have routes to the destination, more route replies will be sent back to the requesting node. This in turn implies that there is a better chance of the source seeing a reply for a route that is close to optimal. In order to increase the number of nodes that a route to the destination, we borrow some techniques from DSR[2].

DSR refers to the technique as aggressive route caching. In a nutshell, each node tries to fill its routing table as quickly as possible. Each node puts its network interface into promiscuous mode, so that all packets on the network will be captured. When a node sees a route reply packet go by, it establishes two new routes. The first is a one hop route to the node that sent the route reply packet. The second is to the destination specified by the route reply packet, using the sender as the next hop. Care must be taken to ensure that these new routes are given the appropriate priority. Both routes should be added to the routing table only if they are not in it already, or if they are better than the current route. The one hop route is given a low priority, since the sequence number of the destination is not known. By filling routing tables, more nodes have the possibility of responding to a route request.

### 2.4.3 Interactions with Expanding Ring Searches

A common technique for trying to improve the scalability of a MANET routing protocol is to use an *expanding ring search*. This limits the broadcast range of a route request packet to a certain radius around the source. However, this can also prevent finding a long lived route that is just beyond the search radius. This does not cause incorrect operation, but it may cause the routes found to be less efficient than without expanding ring searches.

## 2.5 Preemptively Replacing Routes

Once the route is chosen, it is inserted into the routing table along with its estimated lifetime. If an accurate estimate of a route's lifetime can be made, it can be used

to predict when the route will fail. Every time a packet is sent along a route, two checks are done. First, is the route due to expire within a relatively small amount of time? Second, is this the node that originated the packet? If both conditions are met, sending out a route request will allow the current route to be replaced by a better one before it fails. This broadcast should set the minimum life field in the request to a value greater than the estimated lifetime of the route that is about to fail. This ensures that the new route is more desirable than the failing route and that the same, failing route is not returned. Having only the node that is the source for packets traveling this route try to replace it ensures that only one node attempts to find a replacement for the route. If this check was not made, every node on the route would flood the network trying to find a new route, which would have the same effect on the routing tables as just the originating node flooding, but with much more network overhead.

This scheme depends upon a reliable estimate of the route's life. If the estimate is wrong, either the route will fail before it is replaced, or route requests will be sent out when not completely needed. If it is known in advance that an original approximation of a route's lifetime is inaccurate, it should be updated periodically. A simple way of doing this is described in the next section.

### **2.5.1 Accurately Determining Route Lifetimes**

Preemptive route discovery requires an accurate estimate of a route's lifetime, or the preemptive route request process will be carried out at the wrong time. In the scheme described above, this depends on getting an accurate estimate of route lifetime when the route is discovered. In a real world situation, this can be difficult to predict due to environmental interference. With a good knowledge of geography surrounding the network, this problem can be reduced, but not eliminated. If the method used in the implementation is known to occasionally produce inaccurate results, it might be necessary to update the estimate of the lifetime of the link at regular intervals. This can be most easily done by sending an Internet Control Message Protocol (ICMP) message to the destination. The destination would then reply with a packet similar

to a route reply packet. This packet would be updated at each relaying node so that once it gets to the source node, it has a new estimate of the route's lifetime.

# Chapter 3

## Implementation

### 3.1 Architecture

In order to gather data on how the modified protocol performed, it was implemented under the *ns* network simulator[8], with wireless and mobility extensions from the CMU Monarch Project[4]. However, because this thesis was done in an industrial setting, a project goal was to have the code easily portable from a simulation environment to a real implementation. This was accomplished by first writing the implementation, being careful to use as few kernel calls as possible, and keeping the functions responsible for starting the routing daemon separate from the rest of the code. The only calls into the operating system are those dealing with packet reception and transmission, as well as those that update the routing tables. Then, as shown in figure 3-1, the kernel calls and the daemon start up functions were removed, and replaced with a layer that mediates communication between *ns* and the routing code. This provides the same functionality to the routing code as the Unix kernel.

In the *ns* environment, there is no distinction between the routing code and the code that forwards application packets. Under BSD Unix[21], the routing code is usually run as a daemon in user space. Routing application packets along the appropriate path is handled exclusively by the kernel. When *ns* delivers an application packet to a node, it first passes it to the translation layer. In addition to providing kernel services to the routing code, the translation layer deals with incoming appli-

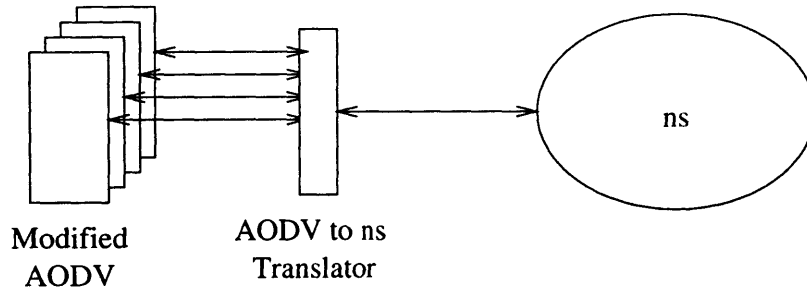


Figure 3-1: Figure showing the overall design of the project. The translator block is designed to be easily replaced with a set of kernel calls.

cation packets. When it receives a packet, it checks to see if there is a route for it in the routing table. If so, it sends it to its next hop destination. If there is no route, it queues the packet, and asks the routing code to find a route. Every time a new route is added to the table, the translation layer processes all the packets in the queue for that destination. These queues are not allowed to grow beyond a certain length and queued packets are dropped after a fixed amount of time. If this was not the case, an unreachable node in the network would cause a large amount of packets to build up in a queue.

## 3.2 Dependencies

As with any routing protocol, this code depends on other features being present in the system. The next two sections describe what features in the simulation this implementation depends upon.

### 3.2.1 Protocols

The implementation is built on top of two protocols in the simulator, both of which are present in standard operating systems. At the lowest level, *ns* provides the IEEE 802.11 medium access protocol[5]. This is well known specification for both the physical layer (PHY) and the medium access control layer (MAC). It provides a set of features ranging from how the transceiver sends bits to collision avoidance. In this scheme, each packet is preceded by a Request-to-Send / Clear-to-Send exchange, and

all unicast packets are acknowledged at the MAC layer.

Nodes in the simulation have both IP and MAC addresses. The routing code deals exclusively with IP addresses, letting code lower in the network stack deal with the MAC address. Translation of IP address to link layer addresses is accomplished by an implementation of the Address Resolution Protocol (ARP)[18], which has been used on the Internet for decades. ARP translates IP addresses into MAC layer addresses. There is a slight problem that occurs when trying to use ARP in a MANET. Typical ARP implementations such as the one found in BSD Unix[21], maintain a queue of only one packet per destination. Since AODV acts in an on demand manner, it is possible to have several packets queued waiting for the route reply message to be received. Normally, once the route reply message is received, the routing layer would send all the packets to the ARP layer. To prevent ARP from dropping all but the last of the packets, either the ARP implementation needs to have a larger packet queue, or the routing layer needs to ensure that no more than one packet is sent to ARP for each destination. This implementation uses the latter approach. This allows the implementation to work with standard ARP implementations, which makes integrating it into current operating systems much easier.

### 3.2.2 Global Positioning System

In the simulation, each node is aware of its position so that it can determine which transmissions it receives. Getting this information to the routing layer is simply a matter of adding the appropriate plumbing. Getting full mission planning information to the routing layer is much more difficult. The only time that a node changes its velocity in *ns* is when it is sent an event from a global event queue. This means that each node in the simulator has no idea where it will be going until the simulation tells it to change direction. The global event queue can be very long and is implemented as a linked list, making searching for a particular event inefficient. To counter this problem, the simulation opens a socket connection to an external program that knows the movement pattern of all the nodes in the simulation. When a node requires its mission planning information, it queries the external program, which responds

with the appropriate data. This closely approximates what would happen in a real system, where the routing daemon would be completely separate from the path finding software.

### 3.3 Protocol Variations

Four different sets of routing code were tested. As a control, an unmodified version of AODV was used. The first, simplest modification adds current position and velocity information to each routing packet. This information is used to estimate the lifetime of both forward and reverse routes, and to pick routes based on a combination of lifetime and hop count (see §2.4.1). The first request for a route has the minimum lifetime field set to ensure a reasonably long lived route is returned (see §2.3.2). It also includes the aggressive route caching features described in §2.4.2 to increase the number of route replies, as well as to lower the need for route request broadcasts. This set of modifications is named `UIOP` in the following discussions. The `PATH` set of modifications includes every modification present in `UIOP`, but replaces the position and velocity information with mission planning information for the next four waypoints. This allows nodes to more accurately calculate the lifetime of a route. The last modification, `PRE`, includes everything from `PATH`, but also sends out preemptive route request packets as described in §2.5. All four sets of code share the same base. Each modification set merely turns on new code paths to implement its changes. This ensures that no one set of modifications has an advantage due to a difference in the common part of the code.

An important factor in how routing code performs are the various constants used in the code. Table 3.1 shows the settings used in all four code sets. Most are identical to those used in [3], which compared AODV, DSDV, DSR, and TORA. The main differences between the standard `AODV` code and the others is that new routes are given an expiration time based on their predicted lifetime, and the first route request for any destination is given a minimum required lifetime.

The last entry in the table refers to when preemptive route requests are sent. Only



	<b>AODV</b>	<b>UIOP, PATH, PRE</b>
Lifetime of a route taken from a Route Reply	600 s	Estimated lifetime
Times a request is retried	3	
Time route requests' broadcast ids are cached	3 s	
Initial expiration time of reverse routes	3 s	Estimated lifetime
Minimum life for first route request	N/A	15 s
Minimum life for all remaining route requests	N/A	0 s
Threshold lifetime for preemptive replacement	N/A	4.5 - 5 s ( <b>PRE</b> only)

Table 3.1: Constants used in the implementation

when the estimated lifetime of route is within the given threshold is a preemptive route request sent. This constant range is set strictly to prevent unneeded preemptive route requests. The danger is best illustrated in the following example. A node receives an application packet for which it has no route. It queues the packet, and sends out a route request. Several nodes respond to the route request. The first response it receives is a route with a low lifetime. It sends the packet on that route, and sends out a preemptive route request, because the route it just used is about to expire. Had the node delayed the preemptive route request slightly, it would have received a much better answer to its original query. This strict threshold might prevent some useful preemptive route requests from being sent, but it also reduces unneeded routing overhead.

# Chapter 4

## Results

While testing any mobile ad-hoc routing protocol, it can be difficult to model an environment so that meaningful results can be achieved. This chapter attempts to show several results about the changes to AODV discussed in the previous two chapters. Most importantly, it shows that packets are still routed properly. This includes delivering packets to the correct destination within a reasonable amount of time and not generating routing loops. After correctness is established, I will attempt to quantify the performance change of the modifications.

### 4.1 Testing Environment

All of the testing of the four protocols described in §3.3 was done using Network Simulator[8, 4]. The *ns* environment comes with extensive facilities for analyzing the performance of the network in many dimensions. Scenarios can either be randomly generated, created by hand, or created with a mix of the two. Each scenario has several parameters associated with it, such as how fast the nodes move, how long they wait before moving to the next waypoint, and the range of their transceivers. This allows for adequate testing of both typical networks, and highly specialized ones. The logging facilities provide a complete picture of the path that each packet travels, including the MAC, routing, and application layers of the network stack at each node.

In addition to providing verbose logging output, *ns* also provides scripts to analyze

the data. These scripts allow statistics such as the number of packets delivered, the number of routing packets created, the number of dropped packets, and the reasons for dropping. Statistics such these are useful for demonstrating correctness, but they do not address the lifetime of the routes chosen. Since the simulator's radio model is deterministic, this can be calculated accurately. To analyze the lifetime of the routes chosen, I wrote a tool that takes in the scenario's movement patterns and the output packet trace file. It builds a dynamic graph of the network in memory, then examines each application packet in the trace file. After determining the path the packet took, it calculates the lifetime of the path. It then looks at the network at that point in time, and determines the longest lived path between those two nodes using dynamic programming techniques. In addition, it also calculates the longest lived path whose length is no greater than the path the packet took. This calculation is needed because it is possible to have a dramatic increase in lifetime from a route with  $n$  nodes to one with  $n+1$  nodes. Since the modified routing code limits the length of the path it chooses, this provides a more accurate description of how well the routing code picks routes. The algorithms used to compute these values are discussed in appendix A.

## 4.2 Packet Delivery Ratio

To determine if the modified code was able to route application packets properly, it was run through numerous randomly generated scenarios. Each scenario consists of 25 nodes in a 670 by 670 meter area. Each node was outfitted with a transceiver with a range of 250 meters. The network size was chosen because I believe it represents a typical number and density of nodes in an ad-hoc network. The geometry was chosen so that at least a few hops are needed to cross the entire field, but also small enough so that the degree of each node was approximately four. With this configuration, most packets could be routed in four hops or less. Also, each scenario had a maximum of ten active connections at any given time, potentially involving 20 different nodes. Each connection was modeled as a constant bit rate stream, sending 512 byte packets four times a second. TCP was not used to prevent its backoff mechanism from lowering the

bandwidth used by each connection. Finally, each scenario was run for 900 seconds, which allowed numerous packets to be sent, yet also allowed the scenario to complete in a reasonable amount of time.

The scenarios were varied over both maximum node speed and pause time. The first velocity, 1 meter per second, is a relatively low speed which is not intended to tax the routing protocol. The more difficult scenarios used an maximum node velocity of 20 meters per second. For each of these velocities, ten different pause times (the amount of time a node spends stationary at each waypoint) were investigated. Pause times ranged in 100 second intervals from zero seconds (constant motion) to 900 seconds (no motion). Finally, for each combination of velocity and pause times, eight different scenarios were generated randomly with those parameters. This lowers the probability of a non-typical scenario at one set of parameters from putting a glitch in the data. In total, this results in  $2$  (speeds)  $\times$   $10$  (pause times)  $\times$   $8$  (samples) =  $160$  different scenarios, each of which was run with the four code sets (**AODV**, **UIOP**, **PATH**, **PRE**).

Figure 4-1 shows the average ratio of packets sent to packets received versus pause time for the four code sets at an average node velocity of one meter per second. In all of the runs, around 32500 packets were sent. As expected, an extremely small number of packets failed to reach the proper destination. Each code set was able to deliver more than 99.8% of all packets. Also as expected, the number of drops converged to zero drops when the nodes were stationary. Upon inspection, network congestion was determined to be the only cause of packets being dropped. Each router has a queue of outgoing packets for each next hop destination. If the router cannot secure the transmission medium within a reasonable amount of time<sup>1</sup>, packets are dropped from the queue. This typically happens when several routes pass either through one node or several nodes close to each other.

The results from the runs where the maximum node velocity was 20 meters per second are shown in figure 4-2. Although the difference is not large in this set of scenarios, the three modified code sets all show a higher packet delivery ratio than

---

<sup>1</sup>This value is set to thirty seconds in all test runs.

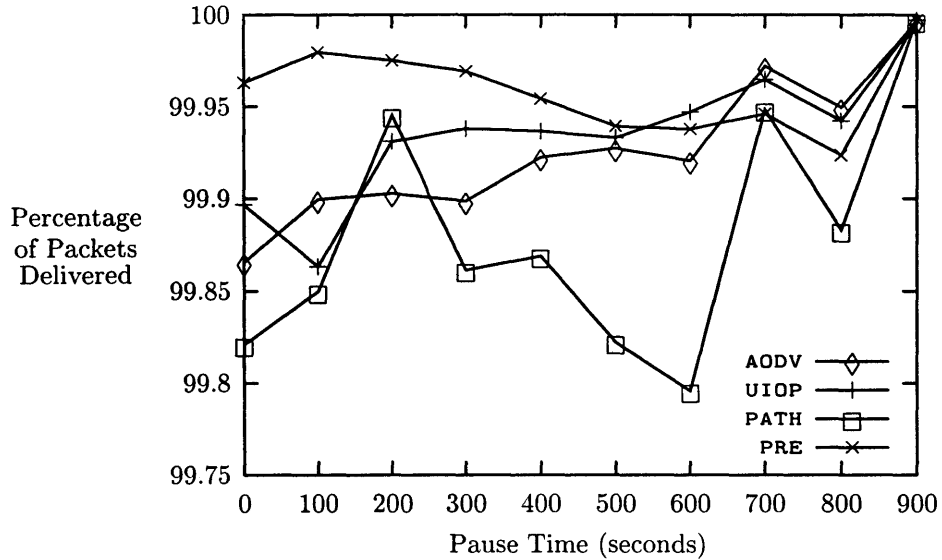


Figure 4-1: Graph showing the average percentage of application packets delivered when the maximum node speed is one meter per second

the control `AODV` code at the left end of the graph. At this speed and low pause time, the average route breaks within a relatively short amount of time. This causes `AODV` to enter the expensive route discovery process more often, which consumes bandwidth, and takes time to complete. All three modifications make those routes that are longer lived, allowing them to avoid the route discovery process.

None of the protocols had difficulty delivering packets under the previous scenarios. To determine how the packet delivery ratio was affected under more stressful conditions, the eight scenarios from above with zero pause time were run with a number of simultaneous connections ranging from 15 to 35. The results are shown in table 4.1. With a lower number of connections, `UIOP` provides a roughly the same packet delivery ratio as `AODV`, while `PATH` and `PRE` provide a slightly better ratio. At the high end, with 35 active connections, the modifications begin to break down, but not as quickly as `AODV`. At this level of bandwidth usage, links appear to break not because the nodes move out of range, but because of severe congestion. The congestion prevents a node from sending a packet over the link. After repeated failures, the node assumes that the link is down. None of the modifications take into account the fact that links might fail for any reason besides being out of transmission range.

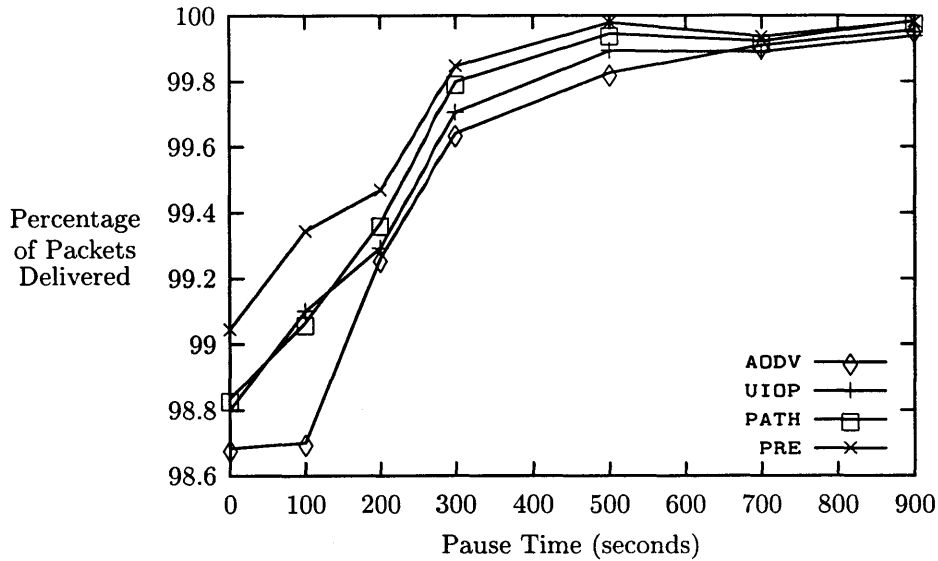


Figure 4-2: Graph showing the average percentage of application packets delivered when the maximum node speed is twenty meters per second

	15	20	25	30	35
<b>AODV</b>	98.81	98.53	98.14	93.82	63.95
<b>UIOP</b>	98.73	98.49	98.27	92.49	65.36
<b>PATH</b>	98.84	98.69	98.52	94.37	66.31
<b>PRE</b>	99.24	99.20	98.98	95.32	71.32

Table 4.1: Table showing the average packet delivery ratio for a varying number of connections

In essence, the modifications can not make a better guess at the lifetime of a link than **AODV** under heavy congestion. However, their lower routing overhead (discussed in the next section) allows them to deliver more packets because less bandwidth is wasted on routing packets. Preemptive route discovery is particularly helpful with 35 connections, because it keeps the needed routes active. Under **PRE**, when a packet is queued, it spends all of its time in the queue waiting for the medium to be secured. Since the other modifications allow routes to expire, packets in their queues first wait for the route discovery process, and then wait for the medium to be secured. This gives **PRE** an advantage because more time is spent per packet trying to secure the medium, giving **PRE** a better chance of transmitting the packet before it times out in the queue.

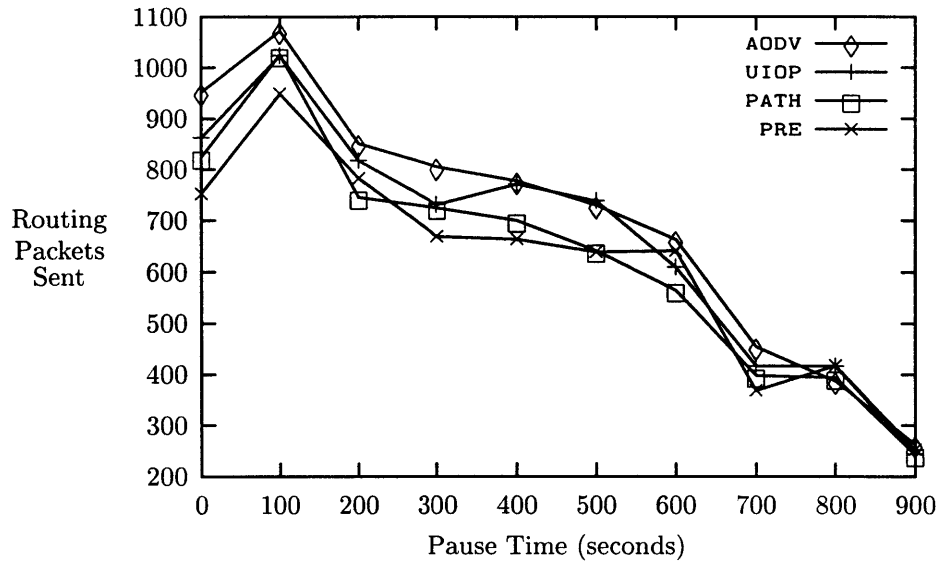


Figure 4-3: Graph showing the number of routing packets sent in each scenario when the maximum node speed is one meter per second.

### 4.3 Routing Overhead

Aside from delivering packets successfully, it is important that the routing protocol does not use too much bandwidth for routing packets. Eventually, this is seen as the packet delivery ratio goes down because the network is choked with routing packets. Figure 4-3 shows the average number of routing packets sent for each pause time when the maximum node speed is one meter per second. Both `UIOP` and `PATH` send slightly fewer packets than the `AODV` control. Because `PRE` is sending out even fewer packets, its preemptive route discovery mechanism is working. For every preemptive request that fails, `PRE` must reinitiate the route discovery phase. This would show up as a higher number of routing packets being sent. In the worst case, `PRE` could potentially send out twice as many routing packets if its preemptive requests always failed.

Figure 4-4 shows the average number of routing packets sent in scenarios when the maximum node speed is twenty meters per second. Note the scale on the y-axis: at this rate of link breakage, `AODV` must send out significantly more routing packets to find and keep active routes. However, all of the modifications reduce the routing overhead by as much as 50%. The `UIOP` code sends out the most routing

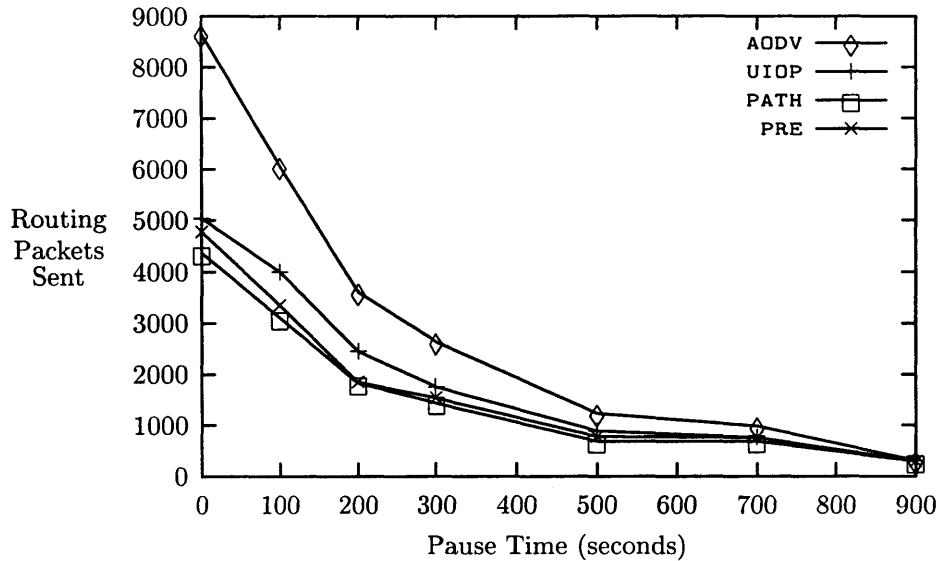


Figure 4-4: Graph showing the number of routing packets sent in each scenario when the maximum node speed is twenty meters per second.

packets of the three modifications, because it cannot predict the lifetime of a route as accurately. But even this weak prediction saves a significant number of routing packets over standard AODV.

The routing overhead statistics for the scenarios with more than 10 connections are shown in table 4.2. In this table, the difference between the modified protocols and AODV is much more apparent. The `PATH` modification almost consistently sends around half the number of routing packets that `AODV` sends. The other protocols perform almost as well. This table shows how the improved prediction mechanism used by `PATH` allows it send out fewer routing packets than `UIOP`, which route lifetime prediction is not as accurate. However, even with relatively inaccurate route lifetime prediction, `UIOP` provides a significant savings in routing overhead as compared to `AODV`. As expected, `PRE` uses slightly more overhead than `PATH`, because its preemptive route discovery occasionally fail, which causes it to restart the route discovery process.

Another key factor in determining the routing overhead is the number of bytes used for routing. All the modifications add octets to the routing packets, but not enough to be significant, if a typical MAC and physical layer specification such as 802.11[5] is used. Frequency hopping spread spectrum (FHSS) is a standard physical



	15	20	25	30	35
<b>AODV</b>	12905	17771	20787	27926	66668
<b>UIOP</b>	7893	10213	12049	18998	49794
<b>PATH</b>	6357	8335	9100	14125	39661
<b>PRE</b>	7295	9132	10402	15139	42200

Table 4.2: Table showing the routing overhead for a varying number of connections

layer specified in 802.11. FHSS headers are comprised of 16 octets which are used for synchronizing the sender and receiver. Next is the MAC header, comprised of 20 to 34 octets, depending on if the ad-hoc routing information is available at this layer. On top of that is the internet protocol (IP) header with 20 octets, and the user datagram protocol (UDP) header with another 8 octets[20]. Finally, the AODV headers add another 20 to 24 octets, for a minimum total of 84 octets in the routing packet itself. But before any routing packet can be sent, the MAC layer must complete a Request-to-Send / Clear-to-Send (RTS/CTS) exchange consisting of 36 and 30 octet packets, respectively. Each unicast packet is also acknowledged at the MAC layer with a 30 octet ACK packet. This equates to a total of 180 octets for every routing packet sent.

The changes in **UIOP** require an extra 12 or 24 octets, depending upon whether the position and velocity are given in 16 or 32 bit quantities. Using these numbers, the routing overhead is about 10% more than with standard AODV. The mission planning information transmitted in **PATH** and **PRE** is harder to quantify. In the worst case, each waypoint will be represented as a set of coordinates and another field representing the velocity used to get to that waypoint. Encoding in this method requires 8 or 16 octets per waypoint, plus a 2 or 4 octet field specifying the initial position. However, most systems that have a use for mission planning information, such as military systems, are tightly integrated. Savings could be realized in several ways through better integration. First, if all the nodes have a common view of the location of the waypoints, waypoint information can be sent in  $n$  bits, where  $2^{n-1} < \text{number of waypoints} \leq 2^n$ . Second, if the nodes travel with a set of fixed speeds, that information can be compressed in the same way. Using these techniques, transmitting mission planning information can use substantially less bandwidth.

	Average Route Lifetime (sec)	Average Difference from Optimal Route (sec)	Number of Routes Used
<b>AODV</b>	13.79	40.37	478
<b>UIOP</b>	23.54	31.87	298
<b>PATH</b>	31.23	28.98	224
<b>PRE</b>	29.72	28.90	295

Table 4.3: Summary of the lifetime test

## 4.4 Path Lifetime

The previous two sections showed that the modifications operated properly under a variety of conditions. In this section, the improvements offered by the modifications will be examined in more detail. This section investigates how much longer the routes chosen by the new code live, and how close they come to longest lived available route. The next section, §4.5, examines in detail the operation of preemptive route replacement, and its success rate.

To observe how much longer lived the routes picked by the modified code are, the code was run on some new scenarios. These scenarios consist of 50 nodes in a 948 by 948 area. This is roughly the same density of nodes as used in §4.2 above. The scenario was enlarged to make the average route contain more hops, and therefore be more probable to break at any given time than a shorter route. The maximum node speed was 20 meters per second, with no pause time at the waypoints. Only one connection was established in this network. This allowed the scenario to complete in a reasonable amount of time and to ease the problem of measuring the path each packet took. Eight scenarios with these characteristics were generated. Each was run with all four code sets for a period of 900 seconds.

Table 4.3 summarizes the results of the test. Whenever the source node picks a new route, its lifetime is evaluated by an omnipotent observer, and compared with the longest lived route of the same hop count. The average new route lifetime is shown in column one of the table. The **AODV** code picks routes that on average, are significantly shorter than the other three modifications. This is because it will accept routes that have extremely low lifetimes, whereas the others will only accept

<b>AODV</b>	1.43
<b>UIOP</b>	1.90
<b>PATH</b>	1.90
<b>PRE</b>	2.35

Table 4.4: Table showing the average difference between the optimal hop count and the actual hop count for each route

short lived routes after they try to discover a route with a lifetime of 15 seconds first (using the minimum life field in the request). The `UIOP` code performs better than the standard `AODV` code, but worse than other two because its predictions are only based on a first approximation of the path of each node.

The second column in table 4.3 shows the average difference between the lifetime of each route, and the lifetime of the longest lived route of the same hop count. The modified code is closer to optimal than `AODV`, but it still clearly does not pick optimal routes. Very few optimal routes are returned because it is very rare that the request will travel an optimal path and be answered. One way to ensure that this happens would be to have every node mark each route request it saw, then forward it to all of its neighbors. Then, instead of dropping each request in its broadcast id cache, it would drop only those that it had marked. Unfortunately, this would cause exponential growth in the number of routing packets, because every possible path in the network from source to destination would be traveled by a route request packet.

As the fourth column shows, picking routes with a longer lifetime results in picking substantially fewer routes. However, the cost of picking fewer routes comes at the cost of some failed requests because the minimum life requirement could not be met. These failures cause extra routing overhead, and delaying the sending of application packets. The `UIOP` code failed a total of 107 times in these scenarios, which is a significant fraction of the number of total routes used. However, `PATH` only fails twice in all eight scenarios. This can be attributed to the better lifetime approximation of the `PATH` code. Similarly, `PRE` only fails a total of ten times. This helps emphasize the importance of accurate predictions.

Table 4.4 shows the adverse effect of attempting to pick longer routes. The first time each route was used, its hop count was measured, and compared to the shortest

hop path between those nodes. The average of these differences is shown for each of the code bases. All three sets of modifications pick routes that are longer than routes `AODV` picks. However, both `UIOP` and `PATH` only increase the average hop count by 0.5 hops over `AODV`. This is because when a route request is sent with a minimum life specified, it is possible that no routes with an optimal hop count have that minimum lifetime. Standard `AODV` code has a similar problem because it does not have an implicit method for picking a route with the shortest hop count. Instead, it assumes that the route request that reaches the destination first will have traveled over the fewest possible number of hops, which is not always the case. Preemptive route replacement does much worse, increasing by an average of one hop over standard `AODV`.

## 4.5 Route Replacement

Figure 4-5 illustrates preemptive route replacement. The scenario is one of the randomly generated scenarios from the previous section. Each point on the graph represents a packet and the lifetime of the route it took. The diagonal lines are formed by a sequence of packets taking the same route. Similarly, every time there is a large jump from point to point, a route discovery process has taken place.

What can be seen in this graph is that very few packets travel along routes that a lifetime of less five seconds, indicating that the preemptive route requests are able to replace current routes before they fail. Preemptive discovery failed in this graph around packets 500 and 1500, where the graph reaches the x axis. It failed because either there was no route with a long enough predicted lifetime, or, more likely, because no route with a long enough lifetime had an acceptable hop count. In all eight scenarios run with `PRE`, preemptive route discovery failed 10 times. In total, 295 new routes were discovered, putting the failure rate of finding a new route before the old one expired at 3.4%.

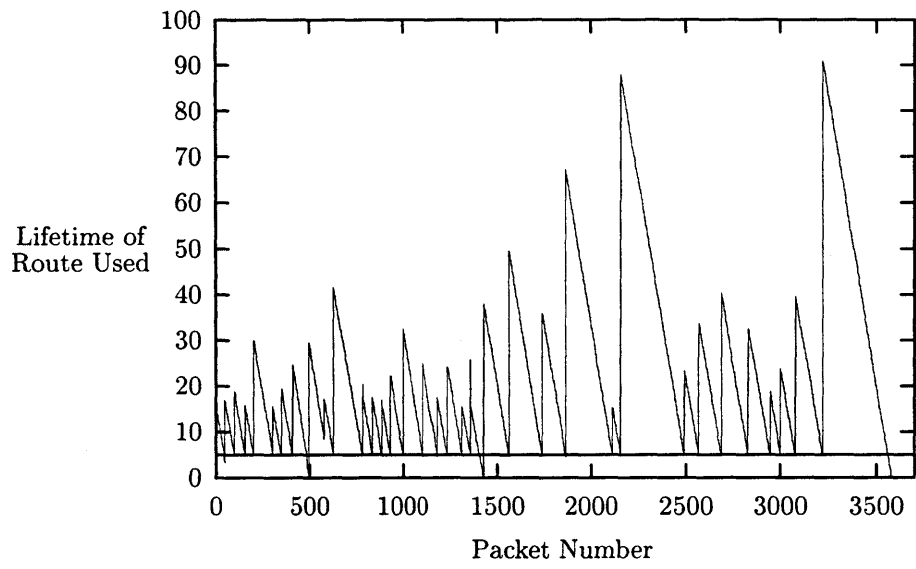


Figure 4-5: Graph illustrating preemptive route discovery

# Chapter 5

## Conclusions

In conclusion, integrating position and velocity information into the Ad-Hoc On-Demand Distance Vector (AODV) protocol can be used to lower the routing overhead in a Mobile Ad-hoc Network (MANET). By using the position and velocity information to predict how long a particular route will stay active, the route discovery process will not have to be initiated as often. Because route discovery is accomplished by flooding the network with broadcast packets, it is an expensive operation which should be avoided whenever possible. Under a variety of different scenarios, the number of routing packets passed ranged around a 50% reduction in the number of routing packets versus AODV. This accomplishment relied upon each node following a predetermined path that could be communicated to other nodes in the network. Such a requirement would be met in a system of autonomous vehicles that are designed to choose a path through their environment and then follow it. If each node is only aware of its current location, using a system such as the Global Positioning System (GPS), a reduction of 30 – 40% in the number of routing packets was observed. Actual routing bandwidth is not reduced by the same percentage because the location and velocity information is piggy-backed on top of the routing packets, increasing the size of the packet by around 10% when only GPS information is exchanged. The overhead used passing mission planning information can be large if an inefficient coding mechanism is used. But, in a tightly integrated system, this information can be compressed to increase the size of each routing packet by about 15%. Overall, the increase in routing

packet size is more than made up for in the reduction of the number of routing packets sent.

Overall, the design goals of the project were met. First and foremost, the modifications reduce the routing overhead required by AODV, as described in the previous paragraph. According to [3], the extra overhead required by AODV is its main disadvantage when compared with Dynamic Source Routing (DSR), another proposed MANET routing protocol. The other project goals, low computation requirements and small routing tables are accomplished by basing the protocol on AODV. The lifetime of a route is calculated in a distributed fashion across the route, which reduces the amount of work that any one node needs to do. Because AODV routes each packet in a hop by hop manner, the amount of information in the routing table is directly proportional to the number of nodes in the network in the worst case, and maintaining the table takes very little computation time.

One of the disadvantages of choosing longer lived routes is that the routes picked usually are longer than the route with the lowest hop count between two nodes. In a set of scenarios where the average shortest route length was just under 3 hops, the modified protocol picked routes that were, on average, 1/2 hop longer. This results in application packets using slightly more bandwidth while being transmitted from source to destination. If the number of application packets being sent is large compared to the size of the network, this technique may cause more bandwidth to be consumed by the extra hops in each route than would be consumed by initiating route discovery more often.

Another potential disadvantage with picking long lived routes in general is that if a shorter route is formed by the movement of the nodes, a protocol that optimizes for longevity will not notice the shorter route until its route dies. For example, suppose two nodes start at either end of a network several hops wide. Using standard AODV, they might establish a five hop route that would be active for seven seconds. The modified protocol might choose a six hop route that would remain alive for 20 seconds. If the nodes are moving closer to each other, by the time the five hop route breaks, there might be a substantially shorter route available. AODV would have found and

utilized this route 13 seconds earlier, when its route broke.

One of the more interesting results is that even a simple approximation of how a node will move can reduce the number of routing packets by 30 to 40% over standard routing. This results were obtained by measuring the current position and velocity of a node, and assuming that it would continue along that path indefinitely. In a real world situation, the network's environment can have an unpredictable effect on the wireless transceivers. This makes it difficult, even given complete information on where each node is going, to predict how long the link between two nodes will remain active. However, this result shows that even an inaccurate prediction can be helpful in determining which route to choose.

Even though the modifications reduce the amount of routing overhead needed, this only helps in cases of moderate congestion. Theoretically, reducing the amount of bandwidth needed by the routing protocol should allow more application packets to be delivered. This was observed in scenarios with moderate congestion. However, under heavy congestion, nodes begin to lose contact with their neighbors because there is no available bandwidth to send packets. Because of this, the node assumes that the route is down, and starts the route discovery process. Because the modification assumes that links only break due to range problems, as congestion increases, the predicted lifetime of each link becomes steadily worse. Under extreme congestion, the modifications essentially make guesses as to how long each route will stay up, and then base their route selection on these completely wrong guesses. In this case, the modifications perform worse than AODV because they tend to favor routes with more hops. Therefore, the reduced routing overhead translates into energy savings and increased scalability to a certain extent.

This thesis also describes a method for replacing routes in a MANET before they fail. It has the same problems with congestion as described above, but under normal network conditions, it will replace the majority of routes before they fail. An accurate initial prediction or a continually updated prediction of a route's lifetime is necessary to make this feature work correctly. With good prediction, it is possible to have fewer than 5% of established routes fail before they can be replaced. Using AODV as a base



for the protocol, it is difficult to reduce this failure rate to zero. Preemptive route replacement is useful in situations where a continuous stream of data is being sent from one node to another. If the sender is sensor with limited memory, this protocol allows it have to have a low latency connection with the intended receiver, preventing it from having to store the data it is gathering while a broken route is in the process of being fixed. However, unless this low latency is needed by the application, this feature should not be used. In general, it sends out more routing packets and picks longer routes than without preemptive route discovery turned on.

## 5.1 Possible Future Work

The concept of a MANET heavily emphasizes that it is self configuring. In its current implementation, the modifications to AODV require that each node specifies a minimum acceptable life for a new route. This parameter depends on the network topology. In a network with long lived routes available, setting it too low results in less than optimal routes being chosen. On the other hand, if it is set to high, the route discovery process fails too often, resulting in a higher routing overhead. Developing a method for setting this value automatically and refining it as the network changes would allow for a maximum reduction in routing overhead.

The modifications typically pick routes that are longer than the ones picked by AODV. A useful addition to the protocol would allow it to “short circuit” some unnecessary hops in these longer routes, and add in the extra hops when they are needed without disrupting the route or adding a noticeable amount of overhead. In essence, the protocol would pick a series of routes, each one differing from the previous by one hop. As the network evolved over time, extra hops could be added or removed as needed to allow the route to remain active. This would confer all the benefits of reduced routing overhead seen in this thesis without the tradeoff of routes that have a longer hop count than needed.

Even though the modifications picks longer lived routes than AODV, analysis shows that it rarely picks the longest lived route from the set of routes with same hop

count as the one chosen. It may not be possible to always choose the longest lived route within a reasonable amount of time and bandwidth. In general, calculating the longest lived route in a graph is a fairly difficult problem. However, any improvements on the methods given here would be welcome. I suspect that significant improvements from the work presented here will only be realized by designing a new protocol that does not rely on AODV. This is because AODV is designed to optimize routes for a low hop count. Changing it to optimize for a combination of longevity and low hop count proved to be difficult.

# Appendix A

## Finding All Pairs Longest Lived Paths

This appendix describes the algorithms used to compute the longest lived paths in a network. Section A.1 describes a specialization of a general framework of path algebras described in [6, 1]. The same algebra is used in §A.2 to compute the longest lived path with an upper limit on its length. Because the implementation limits the hop count of each route, this is a more accurate picture of what an ideal routing protocol would do. Unfortunately, solving the problem of longest lived paths with limits on the paths' length is more computationally intensive than the unlimited version, for reasons that will be described below.

### A.1 Unlimited Length Paths

To solve the problem of computing the longest lived path without regard for its length, we first define an *closed semiring*,  $(S, \oplus, \odot, \bar{0}, \bar{1})$ . In this system,  $S$  is a set of elements,  $\oplus$  and  $\odot$  are binary operations on  $S$ , and  $\bar{0}$  and  $\bar{1}$  are elements of  $S$ . When applied to graph problems,  $S$  is a set of possible values of a link's metric, such as capacity, length, or in our case, lifetime. The extension operator,  $\odot$ , is used to compute the metric of a path, given the metric of each link on the path. The lifetime of a path is the minimum of the lifetimes of the links on the path, so minimum is chosen as the

extension operator. The summary operator,  $\oplus$ , is used to combine the metrics for separate paths. To find the longest lived path, we use the max operator.  $\bar{0}$  and  $\bar{1}$  are chosen to be the identities of  $\oplus$  and  $\odot$ , respectively.

More formally, we choose the closed semiring  $\mathcal{S} = (\mathfrak{R}^{\geq 0} \cup \{\infty\}, \max, \min, 0, \infty)$  to solve our particular problem. It is easily shown that  $\mathcal{S}$  satisfies the following properties of a closed semiring:

1.  $(\mathfrak{R}^{\geq 0} \cup \{\infty\}, \max, 0)$  is a *moniod*:

- $\mathfrak{R}^{\geq 0} \cup \{\infty\}$  is closed under max
- Max is associative:  
 $\forall a, b, c \in \mathfrak{R}^{\geq 0} \cup \{\infty\} \max(a, \max(b, c)) = \max(\max(a, b), c)$
- 0 is an identity for max:  $\forall a \in \mathfrak{R}^{\geq 0} \cup \{\infty\} \max(a, 0) = a$

2. Similarly,  $(\mathfrak{R}^{\geq 0} \cup \{\infty\}, \min, \infty)$  is a moniod:

- $\mathfrak{R}^{\geq 0} \cup \{\infty\}$  is closed under min
- Min is associative:  
 $\forall a, b, c \in \mathfrak{R}^{\geq 0} \cup \{\infty\} \min(a, \min(b, c)) = \min(\min(a, b), c)$
- $\infty$  is an identity for min:  $\forall a \in \mathfrak{R}^{\geq 0} \cup \{\infty\} \min(a, \infty) = a$

3. 0 is an *annihilator*:  $\forall a \in \mathfrak{R}^{\geq 0} \cup \{\infty\} \min(a, 0) = 0$

4. Max is *commutative*:  $\forall a, b \in \mathfrak{R}^{\geq 0} \cup \{\infty\} \max(a, b) = \max(b, a)$

5. Max is *idempotent*:  $\forall a \in \mathfrak{R}^{\geq 0} \cup \{\infty\} \max(a, a) = a$

6. min distributes over max:

- Case 1:  $a \leq b, a \leq c$

$$\min(a, \max(b, c)) = a$$

$$\max(\min(a, b), \min(a, c)) = \max(a, a) = a$$

- Case 2:  $b \leq a, b \leq c$

$$\min(a, \max(b, c)) = \min(a, c)$$

$$\max(\min(a, b), \min(a, c)) = \max(b, \min(a, c)) = \min(a, c)$$

- Case 3:  $c \leq a, c \leq b$  Similar to case 2.

$$\therefore \forall a, b, c \in \mathfrak{R}^{\geq 0} \cup \{\infty\} \min(a, \max(b, c)) = \max(\min(a, b), \min(a, c))$$

7. If  $a_1, a_2, a_3, \dots$  is a countable sequence of elements in  $\mathfrak{R}^{\geq 0} \cup \{\infty\}$ , then  $\max_{i=1}^{\infty} a_i$  is well defined in  $\mathfrak{R}^{\geq 0} \cup \{\infty\}$ . This statement holds if we define max to be the least upper bound of its arguments. For example, let  $a_i = i$ . Then  $\max_{i=1}^{\infty} a_i = \infty$ .

8. Associativity, commutativity, and idempotence apply to infinite summaries. This is true. The properties of max are not changed when it is applied an infinite number of times.

9. Min distributes over infinite applications of max:

$$\min(a, \max_{i=1}^{\infty} b_i) = \max_{i=1}^{\infty} \min(a, b_i)$$

Now, we define a function  $\lambda$ , which, when given two vertices (nodes) in our graph, returns the lifetime of the link between them.

$$\lambda : V \times V \rightarrow \mathfrak{R}^{\geq 0} \cup \{\infty\}$$

$$\lambda(u, v) = \text{lifetime of the link between } u \text{ and } v. \text{ } 0 \text{ if no link exists.} \quad (\text{A.1})$$

By using  $\lambda$ , and the extension operator, we can now define the lifetime of a path,  $p = \langle v_1, v_2, \dots, v_k \rangle$  to be  $\lambda(p) = \lambda(v_1, v_2) \odot \lambda(v_2, v_3) \odot \dots \odot \lambda(v_{k-1}, v_k)$ . In the case of the empty path,  $\varepsilon$ , we define its lifetime to be  $\lambda(\varepsilon) = \infty$ . Since the graph is not guaranteed to be acyclic (in fact, it almost certainly isn't!), we need a way for determining the summary of traveling around a cycle an infinite number of times. Call the cycle  $c$ , for which  $\lambda(c) = a$ . Traveling it zero times will yield a summary of

$\lambda(\varepsilon) = \infty$ , once,  $\lambda(c) = a$ , twice,  $\lambda(c) \odot \lambda(c) = a \odot a$ , etc. We define the summary of an infinite number of traversals around  $c$  to be the closure of  $a$ :

$$\begin{aligned} a^* &= \bar{1} \oplus a \oplus (a \odot a) \oplus (a \odot a \odot a) \oplus \dots \\ &= \max(\infty, \max(a, \max(\min(a, a), \max(\min(a, \min(a, a)), \dots \\ &= \infty \end{aligned}$$

Now that we have the necessary mathematical machinery, we can formulate our problem precisely. We want to compute a matrix that at each element  $i, j$  is the summary (the lifetime of the longest lived) of the paths between them. Call this matrix  $\ell$ . We define it to be:

$$\ell_{ij} = \bigoplus_{i \rightsquigarrow j} \lambda(p)$$

This matrix can be computed using dynamic programming techniques. First, let  $Q_{ij}^{(k)}$  be the set of paths from  $i$  to  $j$  using only the vertices in the set  $\{1, 2, \dots, k\}$ . Using this, define

$$\ell_{ij}^{(k)} = \bigoplus_{p \in Q_{ij}^{(k)}} \lambda(p) \tag{A.2}$$

Since we can compute the base case,  $\ell_{ij}^{(0)}$  using  $\lambda$ , the rest of the matrices can be computed recursively using the formula:

$$\ell_{ij}^{(k)} = \ell_{ij}^{(k-1)} \oplus \left( \ell_{ik}^{(k-1)} \odot (\ell_{kk}^{(k-1)})^* \odot \ell_{kj}^{(k-1)} \right)$$

Since we previously determined the closure of any path to be infinity, and defined

$\odot$  to be min, the closure term can be eliminated from the above expression. Rewriting using our definitions for  $\odot$  and  $\oplus$  yields:

$$\ell_{ij}^{(k)} = \max(\ell_{ij}^{(k-1)}, \min(\ell_{ik}^{(k-1)}, \ell_{kj}^{(k-1)})) \quad (\text{A.3})$$

Now, we have something that can be easily written in a programming language. Translated to (pseudo) C source code, this becomes:

---

```

double **
compute_summaries() {
    /* uses algorithm from CLR 26.4 to compute longest lived routes for links */
    int i, j, k;
    double ***l;
    int nn = numnodes + 1;

    for (i = 1; i < nn; i++) {
        for (j = 1; j < nn; j++) {
            if (i == j) {
                l[0][i][j] =  $\infty$ ;
            } else {
                l[0][i][j] =  $\lambda$ (i, j);
            }
        }
    }

    for (k = 1; k < nn; k++) {
        for (i = 1; i < nn; i++) {
            for (j = 1; j < nn; j++) {
                l[k][i][j] = MAX(l[k - 1][i][j], MIN(l[k - 1][i][k], l[k - 1][k][j]));
            }
        }
    }

    return l[numnodes];
}

```

---

I have left out memory management details in the above code for simplicity. When called, this function first initializes the base case using the  $\lambda$  function, and then begins building longer paths according to the recursion in equation A.3. It returns the matrix that corresponds to including all possible nodes in each path. By looking at row  $i$  and column  $j$  in the returned matrix, we find out the lifetime of the longest lived route

between nodes  $i$  and  $j$ . Since minimum and maximum can be computed in constant time, the above code runs in  $\Theta(n^3)$  time.

## A.2 Limited Length Paths

While the code in the previous section is fairly efficient, it cannot be used when the length of the path must be less than the number of nodes in the network, because at each stage  $k$ , we only use vertices in the set  $\{1, 2, \dots, k\}$ . To remedy this problem, we use a technique presented in [9] that uses the same algebra defined in §A.1. First, define  $C_{ij}^k$  to be the set of paths from  $i$  to  $j$  using any  $k + 1$  vertices. Note the difference from  $Q_{ij}^{(k)}$ , which limited the set of vertices. Now, the matrix  $A$  can be defined in parallel to equation A.2 to be

$$A_{ij}^k = \bigoplus_{p \in C_{ij}^k} \lambda(p) \quad (\text{A.4})$$

However, in this case,  $A_{ij}^k$  can only be computed by raising the base case of  $A$  to the appropriate power in our algebra. In the most naive case, this requires  $k - 1$  matrix multiplications, each taking  $\Theta(n^3)$  time. Since we have shown that  $\oplus$  and  $\odot$  are associative and distributive, the standard result  $A^{p+q} = A^p \odot A^q$  from linear algebra still holds. To compute  $A^k$  efficiently, we decompose  $k$  into base 2.

$$k = \alpha_r \alpha_{r-1} \dots \alpha_0 \quad \text{with } \alpha_k \in \{0, 1\} \text{ and } \alpha_r = 1$$

Then,  $A^k$  can be written as

$$A^k = A^{\alpha_r 2^r} \odot A^{\alpha_{r-1} 2^{r-1}} \odot \dots \odot A^{\alpha_0}$$

which can be done with  $N(k) = r - 1 + \sum_{i=0}^r \alpha_i$  matrix multiplications by first



computing  $r-1$  powers of two of  $A$ , then selecting the appropriate ones and multiplying them together. The code that implements the above ideas is shown below.

---

```

void
matrix_mul(double **a, double **b, int n, double **c) {
    /* sets C=A*B dimensions of A,B,C are nxn */
    int i,j,k;

    assert(a != c && b != c);

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            c[i][j] = 0;
            for (k = 0; k < n; k++) {
                c[i][j] = MAX(c[i][j], MIN(a[i][k], b[k][j]));
            }
        }
    }
}
10

void
matrix_pow(double **a, int pow, int n, double **ans) {
    /* sets ans = A^pow. dimensions of ans,a is nxn */
    double ***b; /* a^2^n = b[n-1] */
    int tmp = pow;
    int pow2 = 0;
    int i,j;
    double **scratch;

    assert(pow > 0);
    /* figure out the highest power of 2 that goes into pow */
    while (tmp) {
        tmp = tmp >> 1;
        pow2++;
    }
    b[0] = a;

    for (i = 1; i < pow2; i++) {
        /* compute square powers of a */
        matrix_mul(b[i - 1], b[i - 1], n, b[i]);
    }
    /* make scratch the identity */
    matrix_ident(scratch, n);

    for (i = 0; i < pow2; i++) {
        if (pow & (1 << i)) {
            matrix_mul(scratch, b[i], n, ans);
            matrix_cpy(ans, n, scratch); /* copy ans back into scratch */
        }
    }
}
20
30
40

```

}  
}

---

Again, the memory management details have been left out. Each of the  $N(k)$  matrix multiplications is done in  $\Theta(n^3)$ , giving the algorithm a running time of  $\Theta(N(k) * n^3)$ , which is slightly worse than the running time for the unlimited length path algorithm in §A.1.

A potential problem with this algorithm is that equation A.4 considers only paths of *exactly*  $k + 1$  nodes. It is possible that a path with fewer hops will have a longer lifetime. This problem is solved by extending the  $\lambda$  function given as equation A.1 so that for all  $v$  in the set of vertices,  $\lambda(v, v) = \infty$ . With this definition of  $\lambda$ , if a path shorter than  $k + 1$  hops has the longest lifetime, the extra hops can be provided by these infinitely long lived paths.

# Bibliography

- [1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley series in computer science and information processing. Addison-Wesley, Reading, MA, 1974.
- [2] Josh Broch, David B. Johnson, and David A. Maltz. The dynamic source routing protocol for mobile ad hoc networks. Internet-draft `draft-ietf-manet-dsr-02.txt`, Internet Engineering Task Force, June 1999. Work in progress.
- [3] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, New York, NY, October 1998. Association for Computing Machinery.
- [4] CMU Monarch Project, Pittsburgh, PA. *Wireless and Mobility Extensions to ns*, August 1999. Available from `ftp://ftp.monarch.cs.cmu.edu/pub/monarch/wireless-sim/ns-cmu.ps`.
- [5] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*. The Institute of Electrical and Electronics Engineers, 1997. IEEE Std 802.11-1997.
- [6] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*, chapter 26.4. MIT electrical engineering and computer science series. The MIT Press, Cambridge, MA, 1990.

- [7] S. Corson and J. Macker. Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations. Request for Comments 2501, Internet Engineering Task Force, January 1999. Available at <http://www.ietf.org/rfc/rfc2501.txt>.
- [8] Kevin Fall and Kannan Varadhan. *ns* manual. Technical report, The VINT Project, January 2000. Available from <http://www-mash.cs.berkeley.edu/ns/ns-documentation.html>.
- [9] Michel Gondran and Michel Minoux. *Graphs and Algorithms*. Wiley-Interscience series in discrete mathematics. John Wiley and Sons, Ltd., New York, NY, 1984. Translated from the French by Steven Vajda.
- [10] Z. J. Haas. The routing algorithm for the reconfigurable wireless networks. In *IEEE International Conference on Universal Personal Communications record*, Piscataway, NJ, October 1997. IEEE Communications Society.
- [11] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In Tomasz Imielinski and Hank Korth, editors, *Mobile Computing*, Kluwer international series in engineering and computer science. Kluwer Academic, Boston, MA, 1996. Also available at <http://monarch.cs.cmu.edu/monarch-papers/kluwer-adhoc.ps>.
- [12] Youngbae Ko and N. H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 66–75, New York, NY, October 1998. Association for Computing Machinery.
- [13] C. Perkins. IP mobility support. Request for Comments 2002, Internet Engineering Task Force, October 1996. Available at <http://www.ietf.org/rfc/rfc2002.txt>. Updated by RFC2290 [19].
- [14] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mo-*

- Mobile Computing and Networking*, pages 207–218, New York, NY, August 1999. Association for Computing Machinery.
- [15] Charles Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance vector routing for mobile computers. In *Proceedings of the SIGCOMM Conference on Communications Architectures, Protocols, and Applications*, pages 234–244, New York, NY, August 1994. Association for Computing Machinery. Updated version available from <http://www.cs.umd.edu/project/mcml/papers/Sigcomm94.ps>.
- [16] Charles E. Perkins. Mobile IP. *IEEE Communications Magazine*, 35(5), May 1997.
- [17] Charles E. Perkins, Elizabeth M. Royer, and Samir R. Das. Ad hoc on-demand distance vector (AODV) routing. Internet-draft `draft-ietf-manet-aodv-05.txt`, Internet Engineering Task Force, March 2000. Work in progress.
- [18] D. C. Plummer. RFC 826: Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware. Request for Comments 826, Internet Engineering Task Force, November 1982. Available at <http://www.ietf.org/rfc/rfc0826.txt>.
- [19] J. Solomon and S. Glass. Mobile-IPv4 configuration option for PPP IPCP. Request for Comments 2290, Internet Engineering Task Force, February 1998. Available at <http://www.ietf.org/rfc/rfc2290.txt>. Updates RFC2002 [13].
- [20] W. Richard Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley professional computing series. Addison Wesley, Reading, MA, 1994.
- [21] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley professional computing series. Addison Wesley, Reading, MA, 1995.
- [22] L. Zhou and Z. J. Haas. Securing ad hoc networks. *IEEE Network Magazine*, 13(6), November / December 1999.