

University of New Hampshire  
**University of New Hampshire Scholars' Repository**

---

Master's Theses and Capstones

Student Scholarship

---

Winter 2010

# Mobile Undersea Routing Protocol

Michael J. Karell

*University of New Hampshire, Durham*

Follow this and additional works at: <https://scholars.unh.edu/thesis>

---

## Recommended Citation

Karell, Michael J., "Mobile Undersea Routing Protocol" (2010). *Master's Theses and Capstones*. 606.  
<https://scholars.unh.edu/thesis/606>

This Thesis is brought to you for free and open access by the Student Scholarship at University of New Hampshire Scholars' Repository. It has been accepted for inclusion in Master's Theses and Capstones by an authorized administrator of University of New Hampshire Scholars' Repository. For more information, please contact [nicole.hentz@unh.edu](mailto:nicole.hentz@unh.edu).

# Mobile Undersea Routing Protocol

BY

Michael J. Karell

B.A., Quinnipiac University (2005)

THESIS

Submitted to the University of New Hampshire  
in partial fulfillment of  
the requirements for the degree of

Master of Science

in

Computer Science

December 2010

UMI Number: 1489958

All rights reserved

**INFORMATION TO ALL USERS**

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI 1489958

Copyright 2011 by ProQuest LLC.

All rights reserved. This edition of the work is protected against unauthorized copying under Title 17, United States Code.



ProQuest LLC  
789 East Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106-1346

ALL RIGHTS RESERVED

©2010

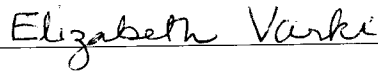
Michael J. Karell

This thesis has been examined and approved.



---

Thesis director, Radim Bartoš  
Associate Professor of Computer Science



---

Elizabeth Varki  
Associate Professor of Computer Science



---

Michel Charpentier  
Associate Professor of Computer Science



---

Steven G. Chappell  
Autonomous Undersea Systems Institute

12 - 23 - 10

---

Date

# Dedication

To Mom.

# Acknowledgments

A number of people have made this paper possible. In particular I wish to thank my advisor, Radim Bartoš, for providing much needed guidance and displaying inconceivable patience. My family and friends for constantly asking me when I would finish my thesis. And lastly, my mother for teaching me never to give up on any endeavor. Ever.

# Table of Contents

Dedication . . . . .	iv
Acknowledgments . . . . .	v
Abstract . . . . .	x
<b>1 Introduction</b>	<b>1</b>
<b>2 Underwater Networks</b>	<b>4</b>
<b>3 Dynamic Source Routing</b>	<b>8</b>
<b>4 Other Work in Ad Hoc Routing</b>	<b>16</b>
<b>5 Mobile Undersea Routing Protocol</b>	<b>20</b>
<b>6 Simulator</b>	<b>28</b>
<b>7 Simulation Results</b>	<b>35</b>
<b>8 Conclusion</b>	<b>44</b>



# List of Tables

7.1	Simulation results: 300 second pause time and 1500 meter transmission range . . . . .	36
-----	---	----

# List of Figures

2-1	Attenuation of acoustic waves in water. . . . .	6
3-1	The Route Discovery process. . . . .	10
3-2	Route Reply Storms. . . . .	12
3-3	An example of DSR Route Maintenance. . . . .	14
3-4	The unidirectional path problem. . . . .	15
5-1	An example of route repair. . . . .	23
5-2	Comparison between Route Discovery and Route Recovery responses to error. . . . .	25
5-3	Comparison between Route Recoveries at 1,2, and 3 hops from the destination and the Required TTLs. . . . .	26
6-1	Simulator class diagram. . . . .	33
6-2	Event Queue Architecture example. . . . .	34
7-1	Simulation results: time to recover from path failure for transmission range 1500 meters. . . . .	37
7-2	Simulation results: time to recover from path failure for transmission range 2000 meters. . . . .	38
7-3	Simulation results: time to recover from path failure for transmission ranges 2500 meters. . . . .	39
7-4	Simulation results: percentage of packets received for transmission ranges 1500 meters. . . . .	40

7-5	Simulation results: percentage of packets received for transmission ranges 2000 meters. . . . .	40
7-6	Simulation results: percentage of packets received for transmission ranges 2500 meters. . . . .	41
7-7	Simulation results: total number of packet transmission for transmission range 1500 meters. . . . .	42
7-8	Simulation results: total number of packet transmission for transmission range 2000 meters. . . . .	42
7-9	Simulation results: total number of packet transmission for transmission range 2500 meters. . . . .	43

# ABSTRACT

## Mobile Undersea Routing Protocol

by

Michael J. Karell

University of New Hampshire, December, 2010

The myriad barriers to underwater communication provide a new set of challenges for network protocols. Routing protocols which operate in underwater ad hoc networks must react quickly to changing conditions without significant increase in packet overhead or congestion. Dynamic Source Routing Protocol provides a framework for accomplishing these goals. In this paper we present the Mobile Undersea Routing Protocol, which implements this framework and enhances upon it. It uses a limited propagating route request which we call a Route Recovery to quickly and inexpensively recover from routing errors. A Java based network simulator was constructed in order to test and compare the protocols. Statistics were calculated based on packets delivered, total transmissions, and time to recover from a route error as measurements of protocol effectiveness.

# Chapter 1

## Introduction

In most computer systems, communication takes place between stationary nodes with propagation delays of only a few microseconds. New technologies have been made available that allow for high bandwidth pipes even in consumer networks. The protocols that have been designed for these networks take advantage of the speed of communication to make it reliable and robust as well. However, these technologies become useless if they are placed in another medium, specifically water. The problem at hand is finding an acceptable communication technology and a set of protocols to facilitate the communication of Autonomous Undersea Vehicles (AUV). We have considered the attributes of both the medium and the network we intend to create in it, and decided on two goals for our protocol. The first is to decrease the time taken to recover from an error. The second is to maintain a comparable end-to-end packet delivery ratio and total transmissions used to deliver packets and for routing overhead. These goals are to be accomplished under varying degrees of network stability which are introduced by a combination of the mobility of the network and the effective range of acoustic communication. We have researched protocols used for land-based mobile ad hoc networks and considered the inefficiencies inherent in each. The protocol we have chosen to base our work on is Dynamic Source Routing (DSR). Preliminary research into prior work has suggested this protocol would be most conducive to our first goal. The Mobile Undersea Routing Protocol (MURP) has been designed with the specific intent of reducing time to adapt to error and maintaining

acceptable levels of quality of service in terms of end-to-end delivery ratio and total number of packet transmissions used for inter-node communication. The rest of the document is organized as follows.

In Chapter 2 of this document we discuss the effects of the underwater medium on communication and the resulting effect on the possible choices of routing protocols. An analysis of the physical medium is presented, including propagation speeds and attenuation of various forms of waves used in wireless communication. The capabilities of AUVs, and how they affect communication, are also discussed in Chapter 2.

In Chapter 3 we discuss the Dynamic Source Routing Protocol as it was originally proposed [12]. The main mechanisms used in DSR are outlined and examined. Route Discovery and Maintenance, the processes by which DSR obtains and maintain source routes between communicating nodes, are covered in detail. Interesting optimizations within DSR that attempt to minimize routing overhead are discussed and critically analyzed.

Chapter 4 outlines various papers that present ways to modify and improve Ad Hoc On-Demand Routing using either Dynamic Source Routing or the Ad Hoc On-Demand Distance Vector Routing Protocol (AODV). Each of these outlines is followed by a critical analysis of the weaknesses of the protocol proposed in the corresponding paper.

Chapter 5 describes the proposed protocol, which adapts the principles of DSR for use in the underwater environment. The Mobile Undersea Routing Protocol (MURP) utilizes a new mechanism called Route Recovery, which provides a fast and inexpensive method to recover from routing errors due to topology change or intermittent failure. Examples are provided to demonstrate the strengths of the Mobile Undersea Routing Protocol in direct comparison to Dynamic Source Routing.

In Chapter 6 we describe the simulator program which was written for this project. The use of object oriented methodologies is discussed and the ways in which they improve the extensibility and ease of use of the program are demonstrated. An emphasis is placed on the principles of polymorphism and inheritance. The program is shown to use a Model View Controller architecture. Event based processing is shown to be used through a simple Queuing architecture. The use of the Spring framework and its dynamic initialization through XML configuration files and object injection is also demonstrated. Design decisions are explained with regards to the implementation of the OSI model for network communication.

Finally, in Chapter 7 we provide results from simulations which show the success of MURP at attaining the goals put forth in earlier sections. We describe the network simulator Java program which was designed and implemented for this task, as well as the details of each experimental setup. We provide the measures used to determine the performance of the protocols. We analyze the results to demonstrate in which ways MURP succeeds in its goals.

## Chapter 2

# Underwater Networks

In this chapter we first examine the obstacles presented by the underwater medium and how they disrupt communication technologies used in wireless networks. We will explore the various wave forms used in wireless communication and how they succeed or fail underwater. In doing so, we will show that acoustic waves are the only form of communication that is viable in Ad Hoc Underwater Networks. We will then examine how this affects the protocol set chosen to control the end-to-end distribution of packets through the network. We will also discuss the capabilities of the Autonomous Undersea Vehicles and how they affect the parameters used in the development of a protocol.

Open air wireless networks generally operate using electromagnetic waves. These technologies are traditionally used for consumer, commercial and municipal networks. Since these networks exist in an unimpeded medium, there is comparatively minimal signal loss and the speed of communication is near to the speed of light. Certain networks are required to function in underwater environments, particularly for scientific or military purposes. In this medium the attenuation, or loss of signal due to absorption, is 45 times the square root of the frequency decibels per kilometer for electromagnetic waves. The absorption of acoustic waves is by contrast several orders of magnitudes lower as shown in Figure 2-1 [13]. However, the propagation delay of acoustic waves in water is much higher in comparison. The speed of sound in water is estimated to be approximately 1500 meters per second [10]. There is also variation



in sound speeds due to changes in temperature, salinity, and hydrostatic pressure. The velocity of sound in water can be more accurately represented by the following empirical formula [2]:

$$c = 1449.2 + 4.6T + 0.055T^2 + 0.00029T^3 + 35S(1.34 + 0.010T) + 0.016z,$$

where  $c$  is velocity in meters per second,  $T$  is temperature in degrees Celsius,  $S$  is salinity in parts per thousand, and  $z$  is depth in meters, which represents hydrostatic pressure. Velocity increases with temperature, salinity and depth. The variation in velocity results in a high degree of unpredictability in the network. Signal multi-path caused by the waves bouncing off of thermoclines, objects, the ocean surface, or the ocean floor result in further unpredictability in the round trip time of a packet. The protocols associated with standard wireless technologies such as IEEE 802.11 are not designed with these factors in mind and are therefore not suitable for use with underwater acoustic modems.

The capabilities of the AUVs have a significant impact on the choices we have made in the creation of communication protocols for them. One of the main attributes to consider is the mobility of the vehicle. The speed at which the vehicle moves determines how quickly the topology may change, which is directly related to the frequency of errors in routing. It also has an influence on the rate at which information about the networks links becomes stale, which affects the caching policy we choose. Were the vehicles to maintain near perfect formation in movement, speed would be less relevant. However, tidal conditions and variations in system performance forestall the feasibility of such a guarantee [6]. In addition, each AUV is capable of being programmed to carry out an individual mission [7]. It is therefore possible that a group of AUVs would move in independent patterns. In a wireless network, the availability of power is another important consideration. Wireless sensor

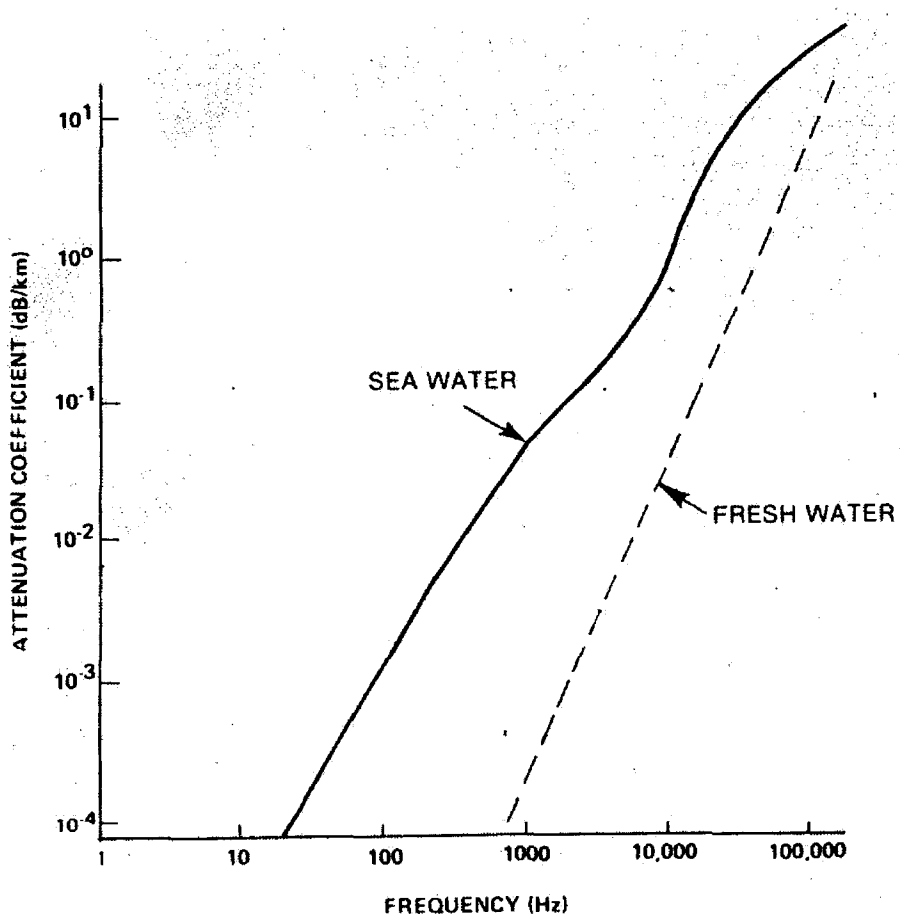


Figure 2-1: Attenuation of acoustic waves in water.

networks tend to focus on reducing overall packet overhead as a means of reducing power consumption [17] [16]. Since the propulsion system, rather than the Network Interface Card and other systems, is the main source of power drain for an active AUV, reduction in total packet overhead is not necessarily a key goal. [5]

## Chapter 3

# Dynamic Source Routing

Dynamic Source Routing (DSR) is a network layer protocol designed for use in ad-hoc networks with mobile nodes and multiple hops between end-nodes. It has been tested successfully on a network composed of laptop computers carried by automobiles in a circular path between two stationary nodes. The mechanisms and descriptions of the DSR protocol that we present and discuss in this chapter are based on the paper by David Johnson, David Maltz, and Josh Broch [8] and the DSR RFC [9]. DSR uses source routing, so packets are transmitted with the entire route record, not just the source and destination. The decision of which path to use through the network is made at the source node, rather than distributed through the network at each node in between the source and destination. In a high latency mobile network, one objective is to minimize power consumption and congestion. DSR is useful in this regard because it attempts to reduce routing overhead by routing on-demand. In an on-demand protocol, the routing table of any node will be empty upon startup and until it attempts to send a packet. The table is filled as the node attempts to communicate with other nodes in the network. There is no periodic route sensing to fill the routing table or to remove stale routes from it. An entry is removed from the routing table if and only if it is discovered to be invalid. The mechanisms for caching and removing routes from the routing table are called *Route Maintenance* and *Route Discovery*.

When a node attempts to send a packet to a node for which it does not have

a route, it initiates a Route Discovery. This is facilitated by a series of messages between the source and destination and the intermediary nodes, which are called Route Requests and Route Replies. The Route Discovery is initiated by broadcasting a Route Request to all nearby nodes. Each successive node sends a new Route Request until it reaches the destination node specified in the Route Discovery. At that point the destination node sends a Route Reply, to which it appends the accumulated route record. Each node receiving the Route Reply will add the route record to its routing table. This procedure is demonstrated in Figure 3-1. The figure shows a node *A* attempting to communicate with destination node *G*. Node *A* initiates the Route Discovery by broadcasting a Route Request packet, which is received at neighboring nodes *B* and *C*. Each of these receiving nodes appends themselves to the cumulative route record and retransmits the packets. Route Requests broadcast from *B* and *D* are received at *C* and *E* respectively. This process repeats and broadcast packets from *C* and *E* are received at *G* and *F*. Since *G* is the destination of the Route Discovery, it transmits a Route Reply back to the source node *A* upon receiving the request. The links in this case are assumed to be bidirectional, so the Route Reply follows the reverse path that the Route Request followed. In this case, *F* also forwards the Route Request to *G*, but it is received after the broadcast from *C*, so that Route Request is ignored as a duplicate.

Since each message involved in the Route Discovery is broadcast, a node may learn and cache any number of routes to multiple destinations as a result of a single Route Discovery. A Route Reply typically follows a route found in the routing table of the destination node. However, if the routing table does not contain a route back to the source, then a new Route Discovery is initiated with the source and destination reversed. To prevent infinite recursion of Route Discoveries, the Route Reply is appended to each Route Request. In cases in which networks have links that are

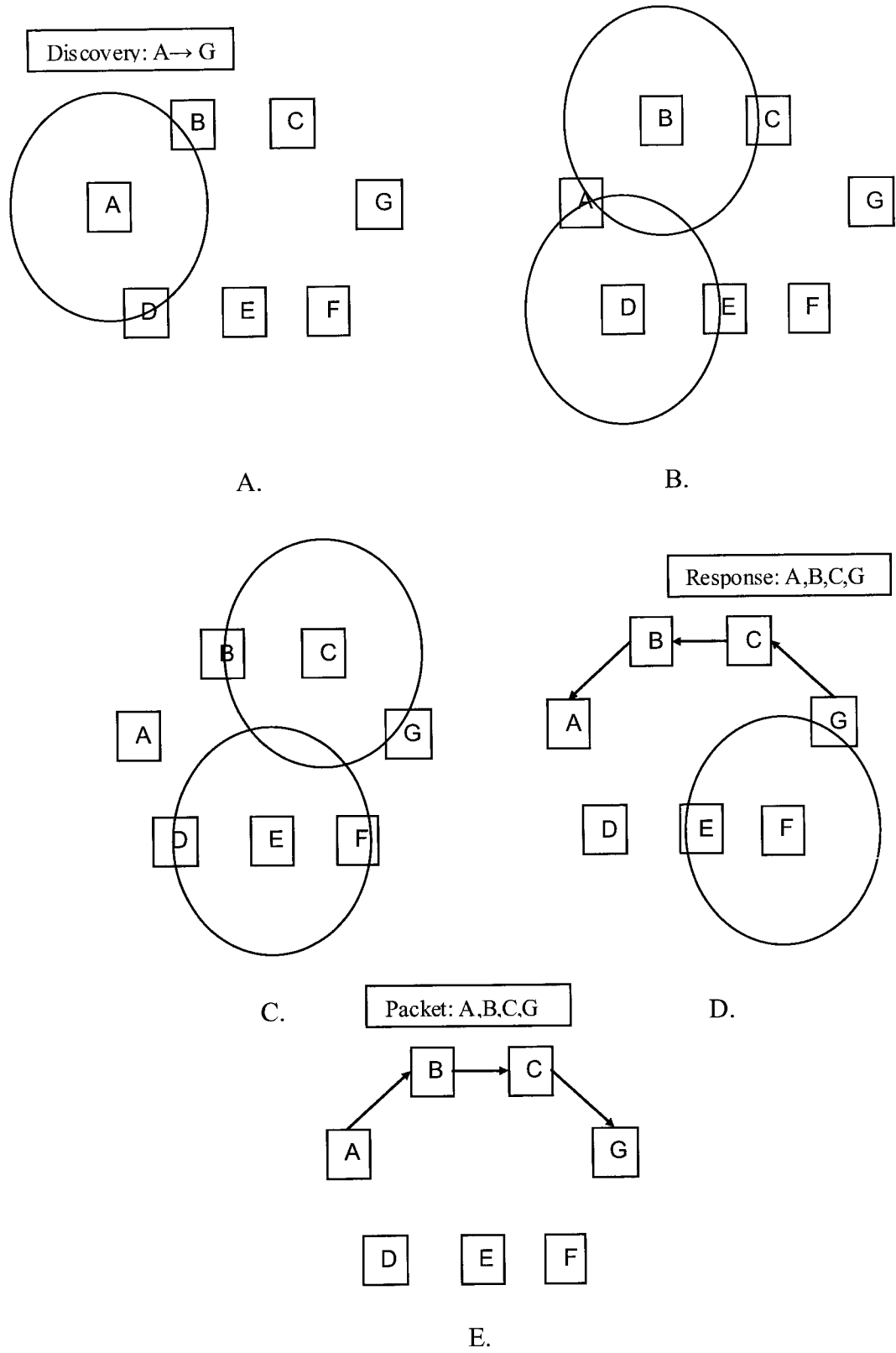


Figure 3-1: The Route Discovery process.

guaranteed to be bi-directional, the destination node may reverse the accumulated route record, cache it, and use it to send the Route Reply. Each intermediate node may also cache a route to previous nodes in the route record. Nodes that overhear the request but are unable to forward it may also cache a route back to the source.

An important issue to consider is that a single route request may result in numerous simultaneous attempts to return a Route Reply. An example of this can be seen in Figure 3-2. Node S requests a route for node *D* and nodes *A*, *B*, *C* and *E* all reply with the routes contained in their route table, which are of varying length. The high volume of Route Replies creates a high probability for congestion problems. In order to prevent this, DSR requires each node to delay its route reply by a random period of time represented by the following formula [8]:

$$d = H * (h - 1 + r),$$

where *H* is a small constant delay, which is recommended to be at least twice the propagation delay of the wireless technology being used, *h* is the number of hops in the route record which is appended to the route reply, and *r* is random number between 0 and 1. In this manner, nodes which reply with longer routes to the target node will send their reply later than those with shorter routes. Each node goes into promiscuous receive mode while it waits for the delay to expire and if it receives a route reply with the same source and destination, but fewer hops, it cancels its Route Reply. While it is necessary to do something to prevent inter-symbol interference, there is a disadvantage to this optimization. Information is lost when the nodes cancel their Route Replies. The routes formed through these nodes, while longer, could still be used in the event that the shorter route is broken. A possible solution is to delay the Route Reply before sending it anyway. This would avoid the congestion while still getting as much information about the network as possible.

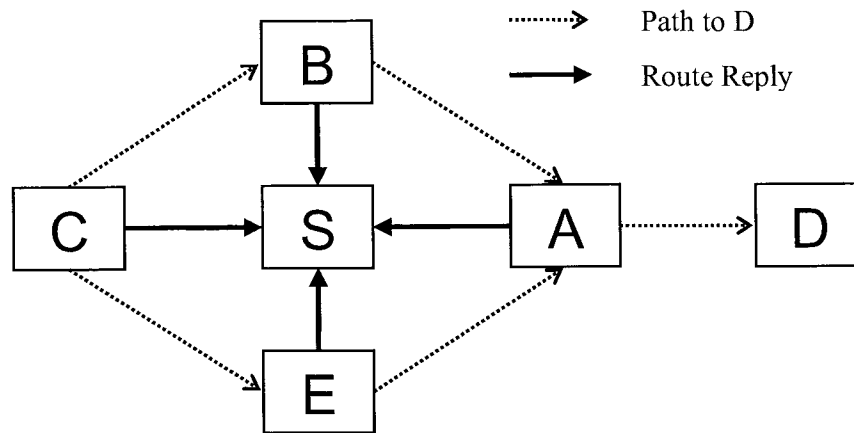


Figure 3-2: Route Reply Storms.

The group of mechanisms by which DSR ensures that the routing table remains current is called Route Maintenance. The basic requirement of Route Maintenance is that each node be responsible for ensuring that any packet it forwards reaches the next hop. The failure of a node to acknowledge receipt of a packet causes the previous node to return a Route Error to the source of the packet. As with Route Replies, the Route Error generally follows a path to the source found in the routing table, but may use a reversal of the current route record if bi-directional links are ensured. There are three ways a node may acknowledge receipt of a packet. One way is for the underlying MAC protocol to provide link-layer acknowledgement of packets. The 802.11 wireless MAC protocol is an example of such a protocol. The second way is for the transmitting node to receive acknowledgement when the next hop broadcasts the packet in its attempt to forward it to the next node in the source route. Lastly, should neither of these options be available, DSR may be configured to require a network layer acknowledgement. This acknowledgement may be sent directly to the previous hop only if bi-directional links are ensured. However, the presence of bi-directional links would in most cases mean that either the previous hop would overhear the packet being forwarded or that a MAC protocol requiring link-layer



acknowledgement is being used. In most cases it will therefore be necessary for any DSR specific acknowledgement to follow a multi-hop path. More important to the efficacy of the protocol is what to do in the case that a route failure is discovered. In DSR, when a node receives a Route Error, it discards the entire associated route and initiates a Route Discovery to attempt to find a new route to that destination. Figure 3-3 shows a simple example of a Route Error and the associated Route Discovery. In the first picture from this example node *A* communicates with node *F* using a source route through nodes *B* and *C*. It is assumed that nodes *D* and *E* were not present when this source route was discovered and therefore are unknown to node *A*. In the second image, node *C* has moved out of range of node *B* and a packet is dropped as a result. As a result, node *B* transmits a Route Error packet back to the source node *A*, which must remove the broken route from its route cache and initiate a rediscovery. The Route Discovery packet is then broadcast in turn from nodes *A*, *B*, *D*, and *E* until it is received at node *F*. Node *F* then transmits a Route Response back to node *A*, which caches the new route and resumes data packet transmission.

The storage and use of routes in the routing table can make DSR a very powerful protocol. A node may cache the route records of overheard Route Replies, Route Requests, or data packets in its routing table and use them later either to forward a packet or return to another node requesting a route to a node on that route record. However, should the link through which the node overheard the information be unidirectional, the route would not function. In Figure 3-4, node *Y* might overhear a Route Reply from node *C* with the route record *C*, *D*, and *E*. Since the link from *C* to *Y* is unidirectional, any attempt by *Y* to use this route will be unsuccessful. Therefore, a node using overheard information must ensure that the link by which it received the information is bidirectional before caching or using it.

Dynamic Source Routing provides basic principles which are very useful for de-

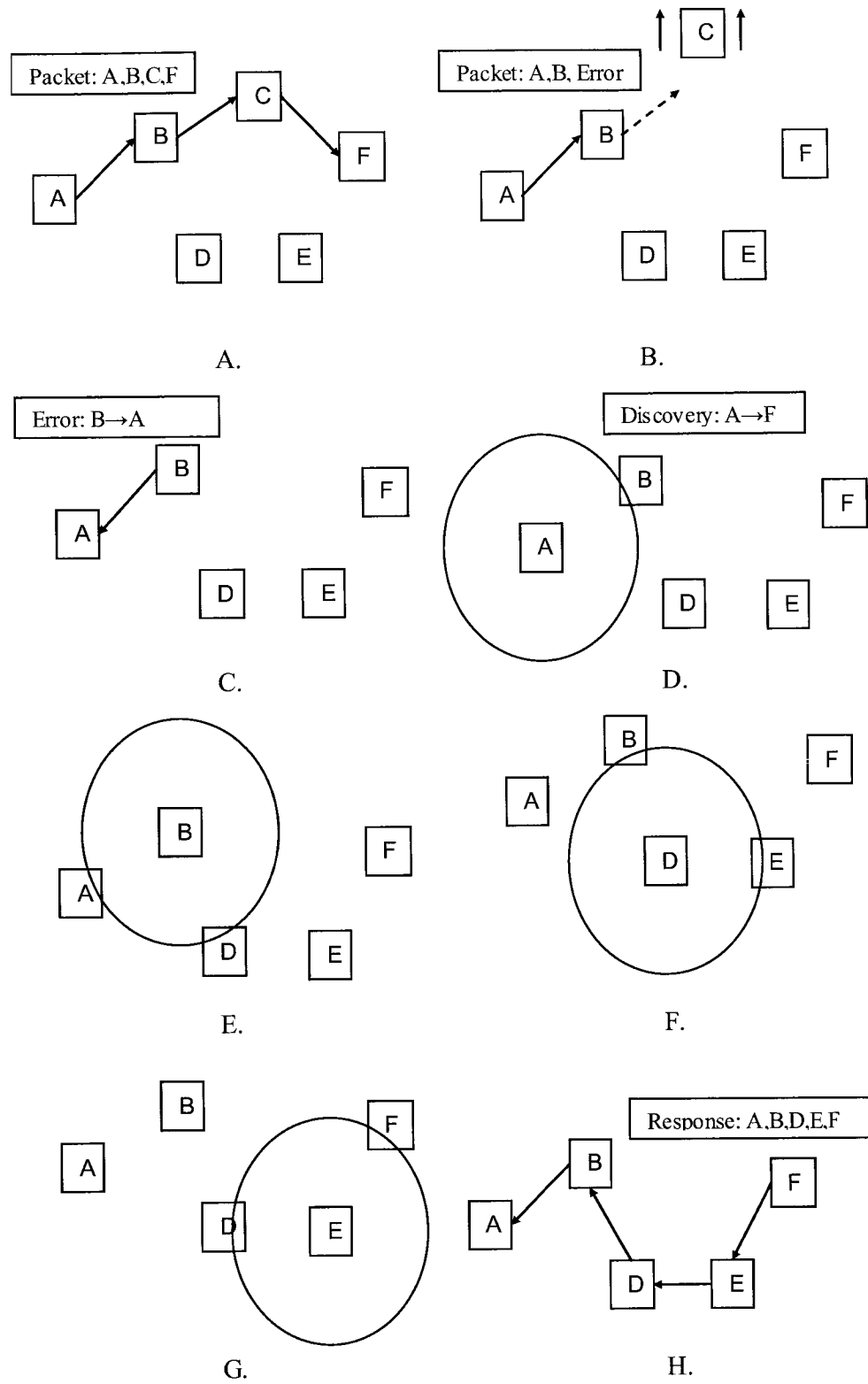


Figure 3-3: An example of DSR Route Maintenance.

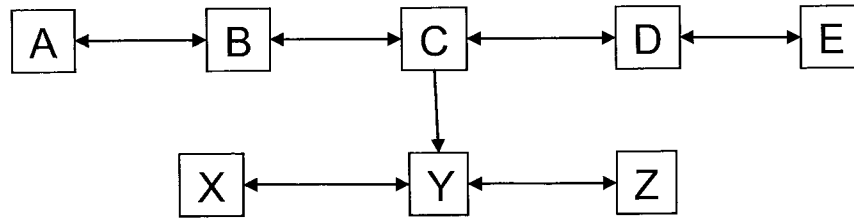


Figure 3-4: The unidirectional path problem.

signing a protocol for use in the underwater environment. The reactive nature of the protocol makes it an ideal starting point given the high latency and unstable topology inherent to an Autonomous Undersea Vehicle network. However, given these same problems, the use of unbounded flooding as defined in Route Discovery is considered to be undesirable. A localized route determination is preferred, particularly for use in Route Maintenance to avoid a complete flooding of the network for each change in topology resulting in route failure. Methodologies to limit flooding in the network are explored in subsequent chapters of this thesis.

## Chapter 4

# Other Work in Ad Hoc Routing

There has been some attempt in other variations of the DSR protocol and other ad hoc routing protocols to handle link failure and subsequent rediscovery in a more intelligent way. The main goal is to lower the routing overhead associated with Route Request and Route Reply messages used in the Route Discovery and Route Maintenance processes. The works discussed in this section all build upon the framework of either DSR or AODV, but some of the optimizations lead to questions about whether the new protocols are suitable for underwater networks. In this chapter we will critically analyze the protocols proposed and discuss their efficacy and possible ways to improve them.

Some papers attempt to divide the networks into partitions in order to create a multilayered network. One such paper proposes the use of Way Point Routing (WPR) [1], in which certain nodes are selected to function as Way Points, which divide the network into segments. In theory any two protocols could be used for intra-segment and inter-segment routing, but in this example inter-segment routing is controlled by DSR, while intra-segment routing is handled by the Ad Hoc On-Demand Distance Vector (AODV) routing protocol [11]. AODV is a distributed routing protocol, meaning that each node contains a routing table with the next address of the next hop needed to reach any destination. The use of both DSR and AODV allows link failure to be handled locally by AODV and Way Point failure to be handled globally by DSR without throwing out the entire route.

Another protocol that partitions the network is the Ad Hoc On-Demand Distance Vector Protocol with Load Balancing (LB-AODV) [14]. This protocol's intention is to reduce overhead by restricting traffic to within groups and designating a group of common nodes to route traffic between the groups. The number of groups is selected so that the difference between the optimal number of mobile nodes (defined as the number of nodes so that each node has 7 neighbors) and the number of mobile nodes that can relay packets from each group is minimal. The number of groups is a tradeoff between the network connectivity and the amount of routing control overhead. [14]. Both LB-AODV and WPR have several similar disadvantages. First, by restricting the paths between nodes, they limit the adaptability of the network. There are fewer available alternative paths in the case of a link disruption. There will also be more congestion over the links between sections and a higher chance of packet loss due to queue overflow. Second, neither paper presents an alternative for the possibility that a section of the network is cut off from the nodes it is allowed to communicate with. In such a case the sectional nature of the protocol would prevent the network from adapting to the new topology and a node could be lost. Speed of recovery from an error is an important consideration of the network and neither of these protocols satisfactorily considers it.

The Congestion Adaptive Routing Protocol (CRP) [15] attempts to reduce routing overhead due to errors and speed up error recovery by discovering and storing multiple routes for a single source and destination. One route is chosen as the primary, while the other routes are used as bypasses in the case of primary link failure. When a link in the primary route fails, the message is sent back along that route until it reaches a bypass that has a destination after the failed link. At that point, the bypass is used as the primary route. If the message reaches the source node, a new primary route must be found. An issue that is not considered in CRP is the way in which the

additional bypass routes are discovered. A single Route Discovery can return multiple routes to the same destination, but some overhead is incurred since it must be assured that multiple routes are found. CRP is also mainly designed to avoid and adapt to congestion. It uses periodic congestion reports to inform the network of which links are congested and route around them using the bypass routes. Since our goal is to minimize the number of packets transmitted, this method of congestion adaptation is not useful to us.

Another technique for enhancing Route Maintenance is Query Localization [4]. When a route fails, instead of flooding the network with Route Discoveries, the old route is appended to each Route Discovery message and a counter and threshold value is set to prevent the message from diverging too much from the old route. Every time the Route Discovery reaches a node that is not on the old route, the counter is incremented. When the counter reaches the threshold value, the Route Discovery is dropped. The idea is that when a node moves it is unlikely that the topology will be very different, so the new route should be somewhat similar to the old one.

The issue with Query Localization is that it makes assumptions about the topology of the network. It will only work if the network is highly interconnected so that possible paths between two nodes differ only by no more than the threshold. If the threshold is too high, the benefits of localization will be minimal. The other disadvantage of Query Localization is that it limits the amount of information gained from the network from each Route Discovery. There is value in knowing routes to as many nodes as possible. One of the strengths of DSR is that a lot of extra knowledge is gained for every packet sent, which can be used for sending future packets.

The protocols outlined in the above sections represent a sample of the improvements to ad hoc routing. While these protocols each have interesting ideas, they were all designed for low-latency environments and their goals are therefore slightly dif-

ferent from ours. Because the propagation delay in acoustic networks is much larger than other factors, the cost of a long packet is not significantly greater than the cost of a short packet. Therefore, while routing protocols for radio frequency ad hoc networks tend to minimize overhead in terms of bits sent, the protocol we have designed for acoustic networks minimizes overhead in terms of number of transmissions.

## Chapter 5

# Mobile Undersea Routing Protocol

The objective of this study is to discern an appropriate routing protocol for communication among fleets of autonomous vehicles in an underwater environment. DSR is a simple protocol which lays the groundwork for the desired application, but it was designed for low latency electromagnetic networks and must therefore be modified to facilitate the high latency of underwater communication. Flooding employed by Route Discovery and Route Maintenance is a major concern in an environment with a high cost per packet in transmission time.

The proposed Mobile Undersea Routing Protocol (MURP) improves upon DSR's Route Maintenance with the goal being quicker recovery from errors as well as fewer total transmissions. The mechanism which we have developed for this purpose is called Route Recovery. It replaces the use of full Route Discovery in Route Maintenance and allows for more localized repair of broken routes.

The Route Recovery mechanism functions as follows. Instead of propagating Route Errors back to the source and allowing rediscovery to occur there, the node creating the error attempts a single Route Request with a time to live (TTL) set based on the number of remaining nodes on the original source route. The theory behind initiating the route repair from the source of the error as opposed to the source of the original route is based on the fact that in a wireless network the connectivity of the nodes is dependent on physical locality. The next shortest path to the destination is likely to pass through or near the error source since it is likely to be in between the



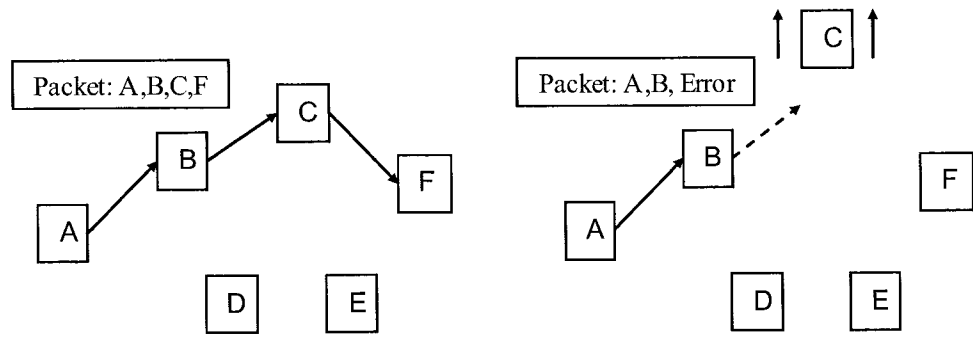
source and destination. The other reason to start the recovery at the error source is that the transmissions used to return back to the original traffic source are costly in the underwater environment. A TTL field in the route header is used to limit the propagation of the recovery transmission to the local area around the error source. A TTL is an integer valued field set in the packet's header which is initialized to a predetermined value and then decremented at every hop which forwards the packet. Once the TTL reaches zero, the node receiving the packet ceases forwarding it and drops the packet. TTLs are used in many networks to prevent an unbounded exponential propagation of every packet. In this case, since packet transmission underwater is expensive, we use TTL to decrease the overhead associated with Route Discovery. However, a full Route Discovery is still used in the case where local Route Recovery is not economical due to an error occurring at great distance from the destination. It is also used to initialize a node's view of the network prior to transmitting data packets.

An example of a Route Recovery is shown in Figure 5-1. This contrasts sharply to using DSR's response to errors using Route Maintenance, as shown in Figure 3-3 and discussed in the prior chapter. As in the prior example, in the first image node *A* has a route to node *F* through nodes *B* and *C* and using this route to send data packets. When node *C* moves out of communication range of node *B*, the route is broken and the packet is lost. However, unlike in the example from the previous chapter, instead of sending a Route Error packet to node *A*, node *B* originates a Route Recovery marking *A* as the original source and *F* as the destination. Nodes *D* and *E* receive this transmission in turn and forward it on until it reaches the destination at node *F*. Once *F* receives the Route Recovery, it sends the Route Response along the accumulated source route, which includes the truncated original source route from *A* to *B* and all nodes which the recovery passed through to reach its destination. At

this point node  $A$  can resume data packet transmissions to  $F$  using this new source route.

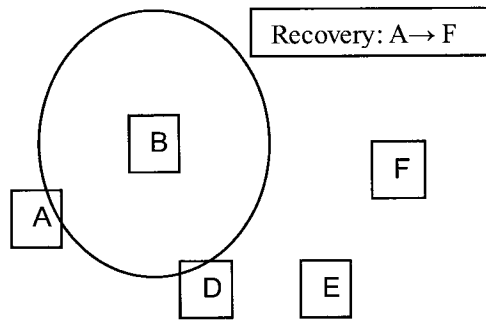
The protocol is configured with a constant minimum TTL to ensure that the request propagates even if very few nodes are left. A constant maximum TTL, above which a Route Recovery is not initiated, is also used to ensure the recovery's scope is not too broad. A recovery bit is set in the error header to notify the source that a recovery was attempted. The source node will then delay its rediscovery attempt by some time large enough to allow the recovery to propagate back first, if successful. If a recovery was not initiated, the node will simply use a full Route Discovery. If the recovery is successful, the destination node will initiate a Route Reply back to the original source. The source should receive this reply prior to attempting the rediscovery and, having reached the lost route, abort the rediscovery and simply transmit a packet along the new route. The intent of this modification is to facilitate quick recoveries in instances where the destination is close by, but allow the source to use Route Discovery to rebuild the topology view with Route Discovery should this fail, or should the failure happen far from the destination.

As shown in Figure 5-2, MURP's Route Recovery mechanism is designed to reduce flooding due to error recovery. In the first image of the diagram, a packet is shown traveling from the source to the destination along the thicker arrows and being between the third hop and the destination. The Route Error transmission is sent back to the source and a Route Discovery is initiated, which floods the entire network with packets attempting to reform a route to the destination. A new route to the destination is found, but at a high cost in both time and packet transmissions. In the second image of the diagram, the same error condition occurs, but a Route Recovery with TTL of 2 hops is used in place of the Route Error and full Route Discovery. The Route Error is appended to the Route Recovery to ensure that the source ceases

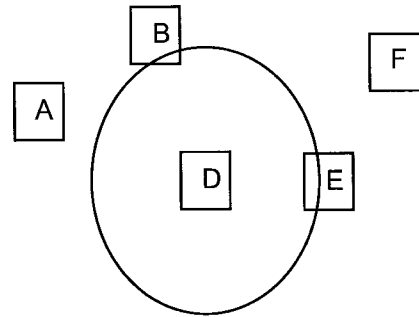


A.

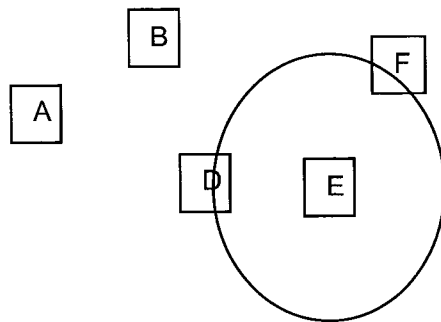
B.



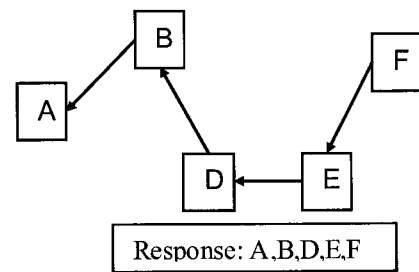
C.



D.



E.



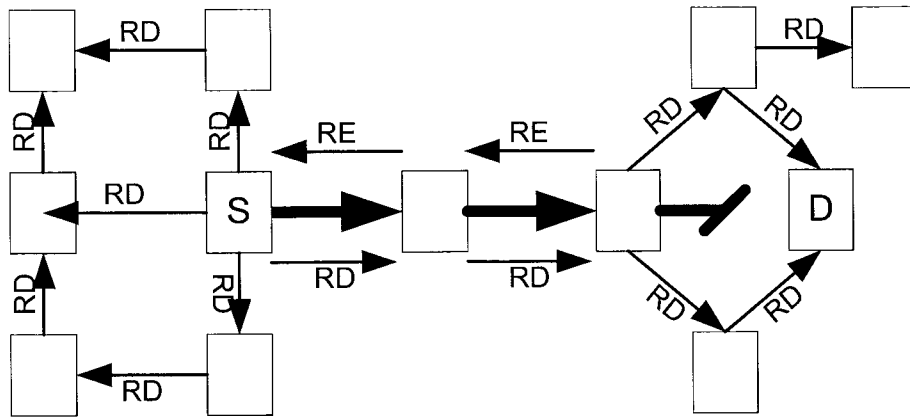
F.

Figure 5-1: An example of route repair.

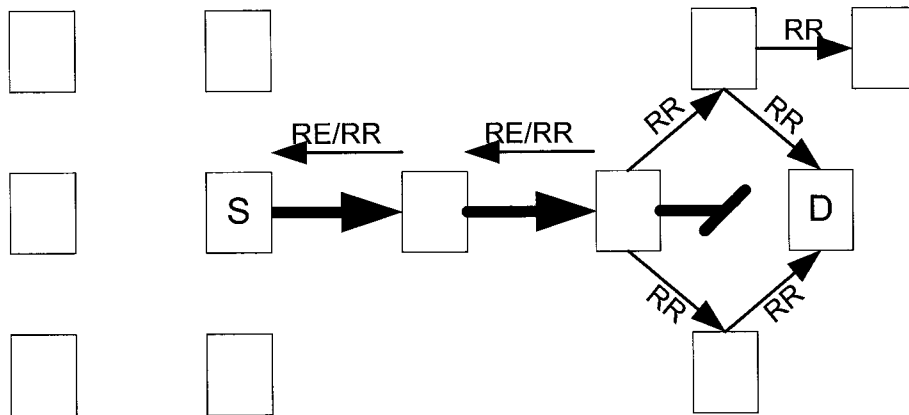
to use the broken route to deliver data packets. The Route Recovery travels to the destination, but is prevented from flooding the entire network with the use of the TTL. It should also be noted that the recovery reaches the destination a full two round trip times earlier in this case due to the Route Recovery being initiated at the error source rather than the original source.

In the above example, one hop was remaining on the original source route and two hops were required to route around the broken link. As shown in Figure 5-3, when a network is assumed to be well connected, the required TTL for a Route Recovery to reach the destination is likely to be one more than the number of hops remaining in the original route. Higher TTL values would be required for loosely connected networks in which few alternate paths exist and each recovery must travel far from the original path to reach its destination. For the experiments in this research, we limit the recoveries to a small TTL with a factor of 1.5 times the number of hops remaining on the original source route, with a minimum of 2 hops to handle the least case and a maximum of 12 hops to further restrict the scope of the recoveries. This we do under the assumption that localized repair is most valuable under minimal scope. In cases in which the calculated TTL would cause the message to propagate to a large portion of the network, we prefer a full discovery to obtain a complete refresh of the topology information.

There are a number of possible measures for success of any routing protocol. From among those we have chosen the three which we feel are most important given the underwater environment. One measure is the total number of transmissions used for packet delivery, including both data packets and packets used for routing mechanisms. In some networks it is desirable to use the routing overhead alone as a data point. However, due to the high cost of each packet transmission in the underwater environment, we choose to use the total packet cost instead of this measure. Another

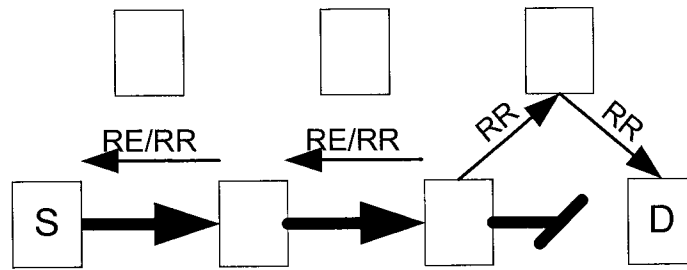


Route Error and Re-Discovery

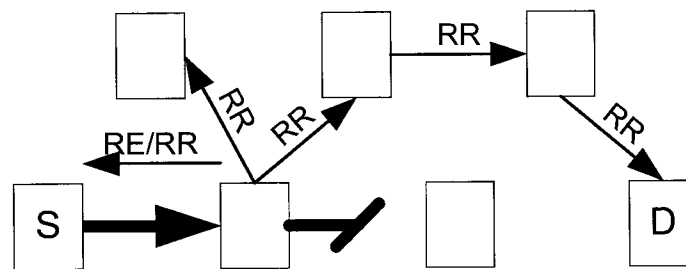


Route Recovery TTL=2

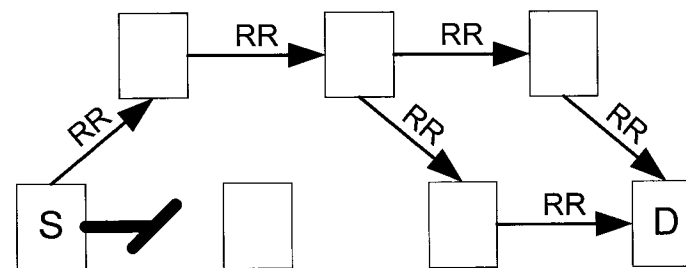
Figure 5-2: Comparison between Route Discovery and Route Recovery responses to error.



Error 1 Hop From Destination  
TTL=2 required



Error 2 Hops From Destination  
TTL=3 required



Error 3 Hops From Destination  
TTL=4 required (assuming no  
reply with cached info)

Figure 5-3: Comparison between Route Recoveries at 1,2, and 3 hops from the destination and the Required TTLs.

possible measure is the end-to-end delivery time. This would measure the difference between the time the packet is transmitted and the time the packet is received at the destination. We reject this measure as well, since the time taken to deliver each packet for which no error occurs would have a strong influence on this number and is unrelated to the capabilities of the routing protocol. The distance between source and destination, which is likely to vary randomly due to the random nature of topologies used, could also have an impact on this measure. We are more interested in the period of time taken by the network to recover from an error. Therefore we measure the recovery time, which is calculated as the difference between the time an error occurs and the time the next packet for that traffic source is received at the destination. This allows us to represent the responsiveness of the protocol to error conditions without respect to the above mentioned variables. A common measure of network performance is throughput. It is the number of successful message deliveries per unit of time. Throughput is often measured in bits per second, but given fixed packet length can be simplified to packets per second. For these experiments, we choose to separate from the time variable and simply consider the percentage of packets delivered over the course of the simulation. This is to show more accurately the reliability of the protocol independently of the speed of delivery. It also prevents the distance between source and destination from having a measurable effect on the statistic. With these measures, we will show MURP to be an efficient and robust protocol.

# Chapter 6

## Simulator

In order to facilitate the experiments required for this project, we created network simulator program in the Java programming language. This program utilizes the latest in enterprise Java technology, the Spring programming framework. Spring provides a number of key components in its framework, the most useful of which is the ability to initialize a program and define objects using XML. The object which facilitates this is the `FileSystemXmlApplicationContext`. When provided with an XML file, this object attempts to initialize all objects defined in the file and set property values given in the object definition. This allows us to modify parameters and, in particular, object types at runtime, without the use of a scripting language such as TCL or a complex property file reading and interpreting process within the Java code.

The second key component utilized in the simulator program is inheritance. While this is one of the most basic Java principles, it is critical for the simplicity of the program. We use a shared abstract super class to define all protocols. This class defines how each implementation of a protocol should function, and following this implementation ensures that the protocol will fit into the simulator code without modification to the main simulator program. Any protocol implementation must have a `send` method and a `receive` method. The simulator stores each protocol and calls its `send` or `receive` method based on events that occur in other protocols or in the simulator's initializing method. The simulator code has references to the specific



protocols used for the simulation. All protocols are stored as the abstract Protocol object, or as in the case of the routing protocol, a more specialized abstract extension of the Protocol class is used to tailor a set of protocols to a specific layer and allow for more layer specific functionality.

The program uses a Model-View-Controller architecture which focuses primarily on the controller portion. The *view* is simplified to a series of logging statements and a print of statistical data at the program's completion. Statistical analysis can be performed both within each protocol and by a separate protocol which watches for events caused by other protocols. In this way statistics can be as simple or complex as the implementor designs. The *model* is maintained in memory and consists of a set of protocols, a set of agents which represent the topology, and packet objects which are generated by the simulator and by each protocol. The relationships of the classes in the model are outlined in the UML class diagram pictured in Figure 6-1. Each packet corresponds to a set of event objects which represent the handling of that packet by a specific protocol at a specific node agent. These events are stored in a priority queue object which is sorted based on the time the event is set to occur. An insertion sort algorithm is implemented as part of the queue and is maintained within the queue implementation. The SimulatorController class runs a loop which pulls the first event from the queue and processes it through the appropriate protocol and as a result generates future events. The SimulatorController class contains references to all protocols, the topology, and the EventQueue class. Each protocol also contains a reference to the topology to allow it to make determinations about the location of nodes and calculate their impact on events generated by that protocol. The time associated with an event is determined by the protocol which generates it. In the case of inter-node communication, an event is generated for every node in range of communication. The group of nodes in range, and the arrival time at each node, is

determined based on the topology model. Any implementation of a topology model must provide an implementation of the NodeAgent class and that implementation must contain a method providing the location of a node at any given time, which is used by the data link layer protocol to determine the time at which a receive event occurs at that node.

The EventQueue class is an extension of the PriorityQueue Java class which implements a comparator to sort its contents based on the time parameter in each Event object. An example of the ways in which events are handled in the queue is outlined in Figure 6-2. In the example, two traffic events, labeled Traffic 1 and Traffic 2, are generated with times 0 and 20 respectively. Traffic event 1 is evaluated first and results in the creation of send event labeled Send 1A and assigned time 0. Since that event preempts the second traffic event, it is executed first and two receive events are generated based on it. The first is labeled Receive 1A and given time of 10 and the second is labeled 1B and assigned a time of 30. The difference is assumed to be based on a difference in propagation delay between the nodes the events represent. Therefore, it is shown in the diagram that Receive 1A will evaluate before Traffic 2, and Receive 1B will evaluate afterwards. When Receive 1A is processed, it generates another send event called Send 1B. Since we assume that the time to process and transmit is negligible in this case, the time assigned to the send event is 10 also. The sequence will continue like this until all events have been processed. In this case we consider only generic send and receive event types, rather than the link layer and route layer send and receive events separately, in order to simplify the diagram. Related send and receive events occurring within the same node will always occur in the order of the layers processing due to the natural order of insert provided by the queue for events with equivalent time values. Each event remains in memory until processed and removed from the queue, at which point it becomes eligible for garbage

collection and will be deleted. The absence of database persistence facilitates quicker processing through reduced I/O time. However, this also results in significant memory usage for each simulation. Simulations for this research used between 1 and 2 gigabytes of memory on average.

The controller handles all interaction between the protocols. Each event contains a type identifier which corresponds to a specific protocol layer and direction of transmission. The controller uses this field to determine what protocol to use and which method to call. To facilitate the above mentioned statistical analysis, a set of event types are also provided which reference methods in the Statistics class to collect data. Each protocol must generate the requisite events for the next layer in the stack as well as any desired statistic events. For example, the protocol must generate an application layer receive event when a packet arrives at its destination and data link send events each time it needs to forward a packet. For the purposes of this project, we have simplified the protocol stack to combine data and link layers and eliminated the transport layer, assuming it to use the simple Universal Datagram Protocol. These protocols could be easily implemented by adding references and corresponding events to the controller and modifying the existing protocols to create the appropriate events. Each protocol implementation should also be complemented by a header class. To avoid interoperability issues, a protocol should only use information from its corresponding header. Cross-layer communication is not currently supported, but will be implemented in future work.

Upon initialization the simulator generates one packet automatically, as well as the corresponding application layer send event, for each traffic agent. The application layer protocol then executes that event, generates the route layer send event for that packet, and creates the application layer send event for the subsequent packet. In our experiments we use a Constant Bit Rate application protocol, such that the

next application send event is set to the current time with an added configurable delay. The event queue's sorted insert ensures that these and all subsequent events are processed in the correct order. The simulator program runs on a loop to easily average the results of multiple experiments for each chosen configuration. It generates a new topology and traffic model for each iteration and tabulates the results in the statistics object. Averages and standard deviations of each statistic item are output after all iterations have completed.

In order to run simulations using this program, the user needs a set of jar files containing simulator source code and other required libraries, a shell script to start the simulator, a configuration file for logging, and a spring configuration file which will define the parameters for the simulation. The required jar files are `spring.jar`, `log4j.jar`, `commons-logging.jar` and the provided simulator source code, which we named `murp.jar` after this project. A `log4j.properties` file must also be included to define the behavior of the logging utility and direct its output. We recommend sending logging output to a file and reserving the standard output for the end of simulation results. The shell script must set the location of each of these things to the class path and then run a Java command to start the simulator. This command must specify a maximum heap size (we use 4096 mb) or else the simulator may run out of memory on larger topologies. The main class to start the simulator is in `com.murp.simulator.controller.SimulatorController`. This class takes the path of the spring config file and the log4j property file as arguments. The log file will print every event in order as it processes, which is mainly useful for debugging protocols. The parameters of the simulation will print to the standard output prior to each run, and statistical results will print after each run for the topology in question. Averaged results for all runs will be printed at the end of the simulation. We recommend redirecting standard output to a separate file for each simulation run. When using one of

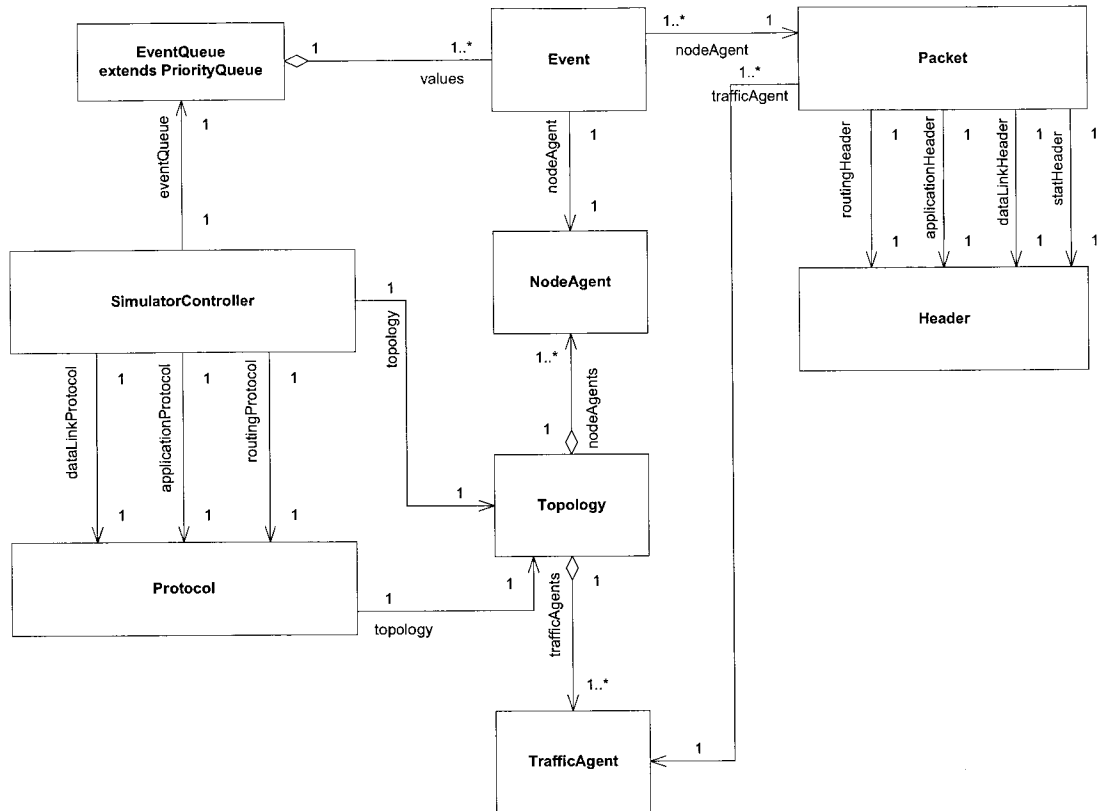
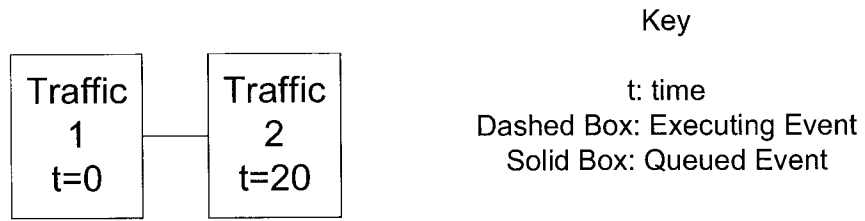
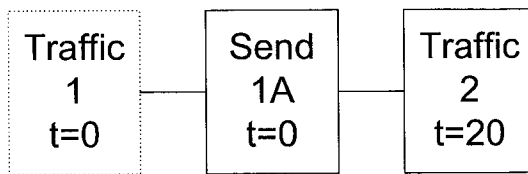


Figure 6-1: Simulator class diagram.

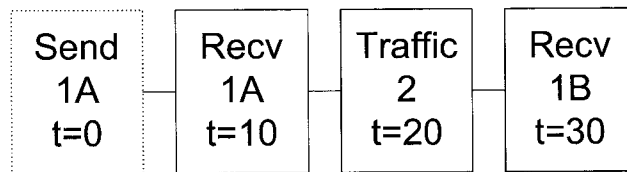
the included random topology generators, such as the Random Waypoint Model, the topology must be configured with the `xmlDirectory` property. This property defines the location to which a copy of generated XML for each topology is saved and will allow the viewing of the topology that is generated for debugging purposes. Currently, if the simulator is configured for multiple runs, only the last XML file will be saved since the naming convention is based on the size and type of topology. In the future an incremental value will be used to save all topologies for each execution of the simulator program.



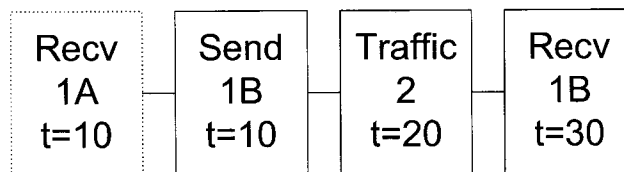
Traffic Generated



Execute Traffic 1 – Generate  
Related Send Event



Execute Send 1A – Generate  
Receive Events



Execute Recv 1A – Generate  
Send Event

Figure 6-2: Event Queue Architecture example.

## Chapter 7

# Simulation Results

In order to test the proposed routing protocol, we use the above described Java application based network simulator to run repeated tests under a variety of conditions. The application's use of the Spring framework for object injection allows run-time configuration changes, which are used to incrementally change the conditions of the simulation. We have explored the use of the NS-2 simulator, but it does not contain a native underwater physical layer and extensions which provide that functionality are not well supported. The physical layer of our simulator uses a constant propagation speed of 1500 meters per second [10] and a simple exponential loss model as a function of the distance between the nodes. The link layer is simple and is combined with the physical layer. We do not explore media access control in this research. The transport layer is implicitly defined to be UDP. The application layer is CBR which is defined to produce a packet at specified intervals for each traffic agent in the simulation.

The topology model used for simulations in this project is the Random Waypoint Mobility Model [3]. This is the simplest model which provides the moment of stability between movements and makes sense in an underwater environment. Other models, such as the Random Walk Mobility Model, its probabilistic version, and the Random Direction Mobility Model only allow continuous movement and lack the temporary stability desired for these experiments. The City Section Mobility Model is disregarded for the obvious conflict of environment, and the Gauss-Markov Mobility Model is ignored because it utilizes extremely complicated mathematical formulae.

Table 7.1: Simulation results: 300 second pause time and 1500 meter transmission range

Protocol	<i>DSR</i>	<i>MURP1</i>	<i>MURP2</i>
Fraction Received	0.619	0.621	0.555
Recovery Time (s)	156.964	144.658	150.8622
Transmissions	143492.06	154050.88	127289.66

The Random Waypoint Mobility Model makes use of a series of randomly generated positions, and a universally defined pause time to define the movement of nodes in the network. The pause time is the amount of time at which a node pauses at each waypoint before moving towards the next one. Pause time is used as a measure of the volatility of the network. In general, the number of errors found in a network has an inverse relationship with the pause time defined for that network. A network with pause time of 0 contains continuously moving nodes. The array of positions, pause time, and the node velocity are used to calculate position at any given time. When the link layer protocol attempts to determine if a packet would be received at a particular node, it calculates the position of the node at the time the packet is transmitted to determine if it is in range of the transmitting node. The transmission delay is then calculated based on the above cited propagation speed and the distance between the two nodes. The delay is then added to the time of transmission to determine the receipt time. This calculation is repeated for each node which is determined to be in range. This version of the model ignores the change in position of the receiving node after the transmission of the packet in order to simply the formula used. An exact formula would require calculation of the intersection of a line and a three dimensional



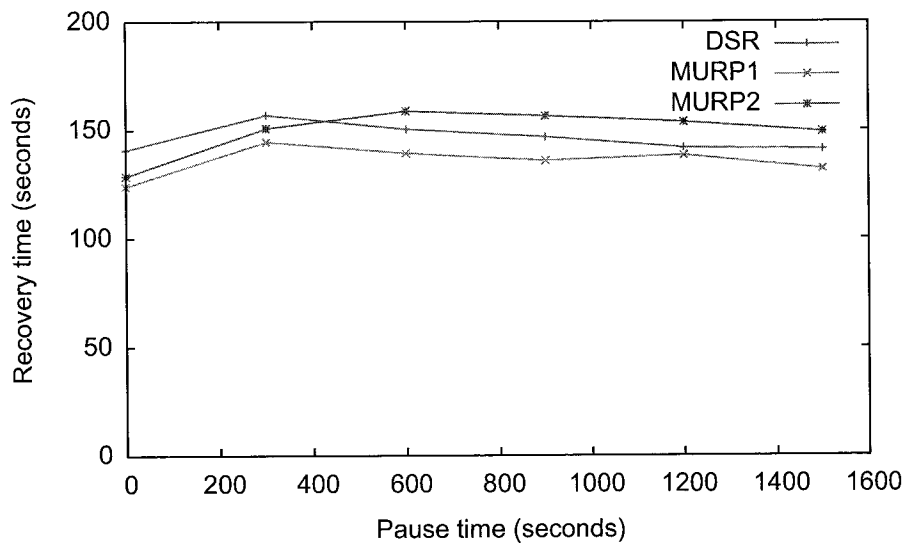


Figure 7-1: Simulation results: time to recover from path failure for transmission range 1500 meters.

cone which would represent the node and the omnidirectional packet transmission respectively.

We have implemented versions of both the standard DSR protocol as described in the DSR RFC [9] and the MURP protocol described in this thesis. Other protocols described in our research will be implemented as part of future work. The success of the MURP protocol is measured in direct comparison to results achieved by the DSR implementation. The DSR implementation was created first and then used as a basis for the MURP implementation. In this way we mirror the design process in the implementation. This ensures that as few as possible additional variables are present in the implementations, which may skew results.

Table 7.1 displays simulation results for each protocol implementation under specific conditions. The chosen settings of 300 seconds for pause time and 1500 meters for wireless transmission range represent a significant impact of mobility. For these results, MURP improves upon recovery time, while maintaining approximately equiv-

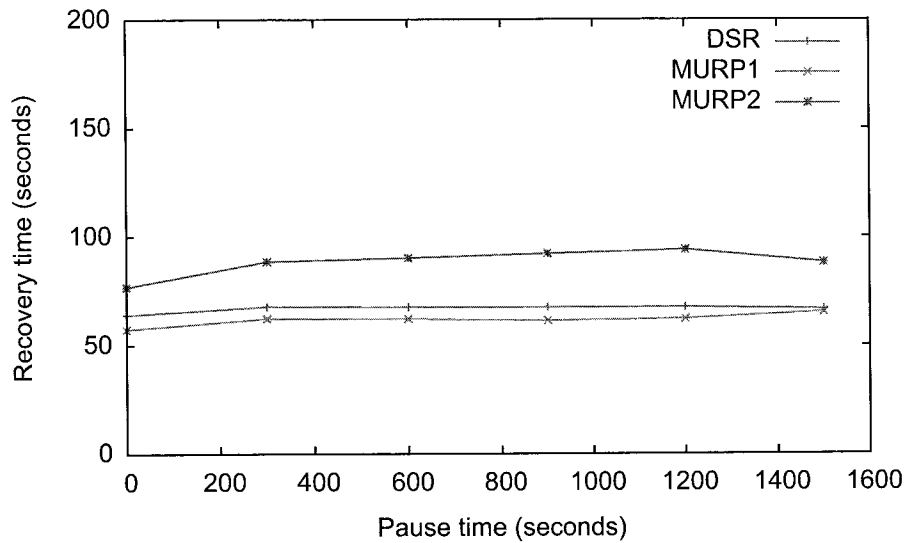


Figure 7-2: Simulation results: time to recover from path failure for transmission range 2000 meters.

alent packet delivery at a cost in overhead of about 11000 packet transmissions.

We have included several graphs to show how each measure is affected by the degree of mobility in the topology as measured by pause time. Each graph shows one of the three chosen measures in relationship to pause time for each protocol at a given allowed range of communication. A comparison of each line in the graph to another line at a given pause time value shows the performance of one protocol respective to the other for the measure represented by that graph. All simulations were run with random 5 by 5 topologies with 10 independent traffic agents moving in 1000 meter square cells at 2 meters per second for 500,000 seconds. The pause time is the delay in seconds a node waits before moving from each waypoint. Each data point represents the average result from 50 independent runs of the simulator. Each run of the simulation tested 1000 packets sent from each traffic agent at 50 second intervals for a total of 10,000 packets. The Random Waypoint Model topology is generated at the start of each run and produces waypoints for up to 500,000 seconds,

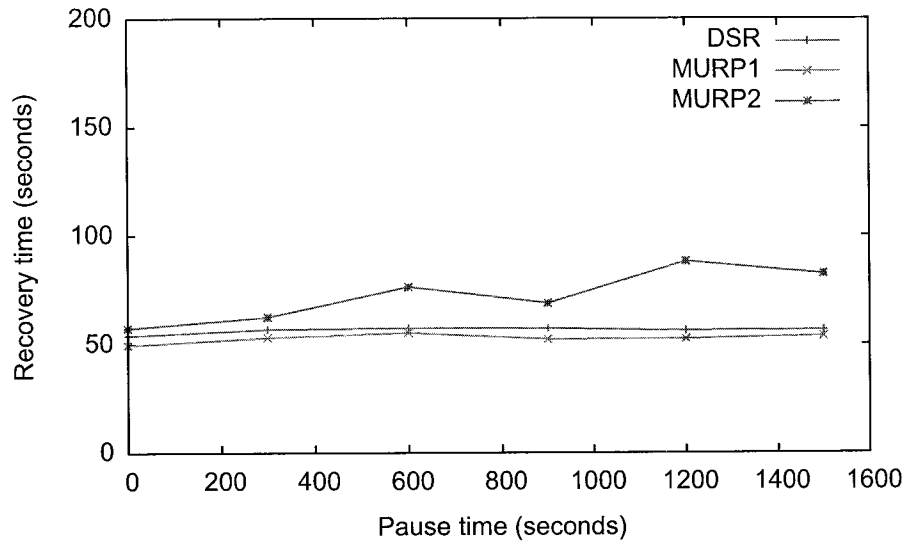


Figure 7-3: Simulation results: time to recover from path failure for transmission ranges 2500 meters.

which is well above the amount of time required for simulation. It is first notable that both protocols struggle to deliver packets with a 1500 meter transmission range. These experiments were intended to stress the network to see how the protocols would react. Also bear in mind that DSR was not intended to be responsible for end-to-end delivery and nor is MURP. The application or transport layer would be responsible for ensuring lost packets are retransmitted. The results for packet delivery shown in Figures 7-4, 7-5 and 7-6 are only intended to show the general reliability of the protocol under given conditions.

Each of the graphs shown in this chapter contains three lines. A line in each is drawn to represent the simulation results for DSR, the final implementation of MURP, which is labeled MURP1, and a second implementation of MURP, which is labeled MURP2. This second implementation was created with the intention of reducing the routing overhead by forcing the source node to send a time delayed Route Discovery upon receiving a Route Recovery with an attached Route Error for which

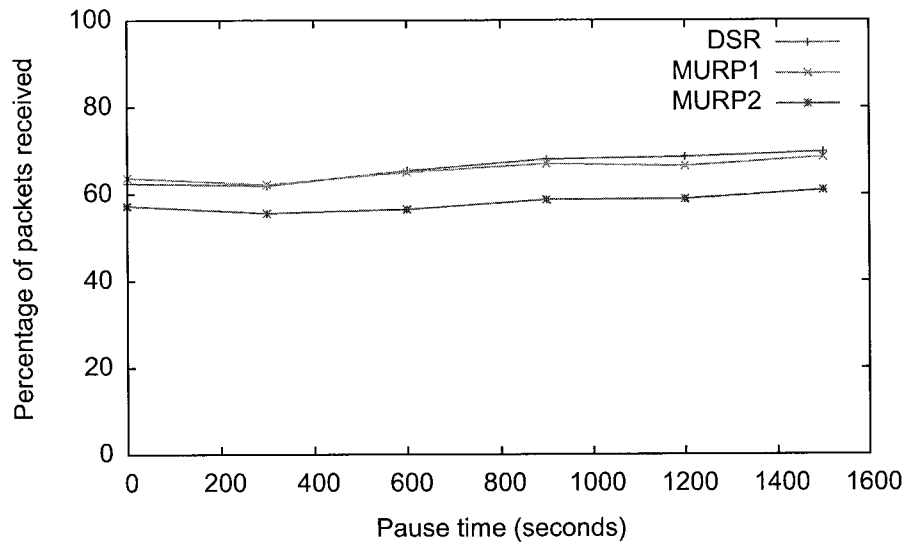


Figure 7-4: Simulation results: percentage of packets received for transmission ranges 1500 meters.

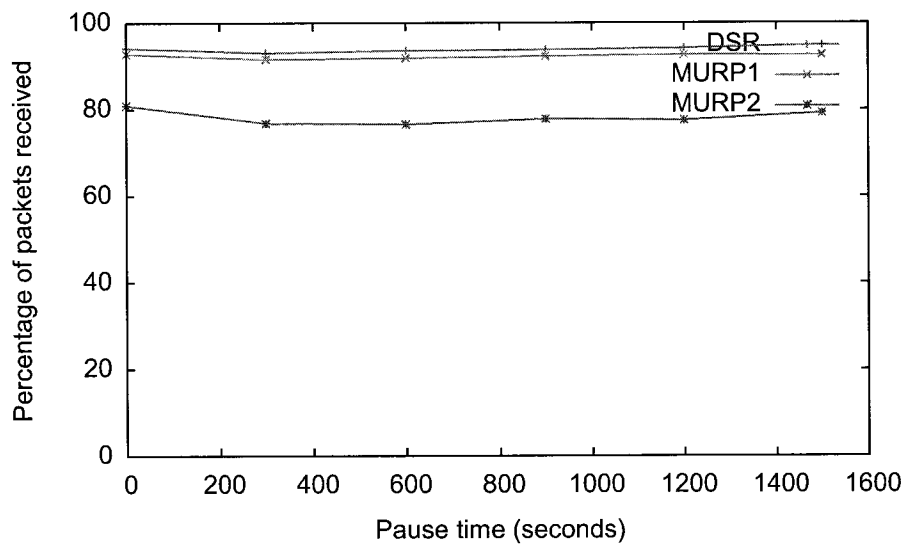


Figure 7-5: Simulation results: percentage of packets received for transmission ranges 2000 meters.

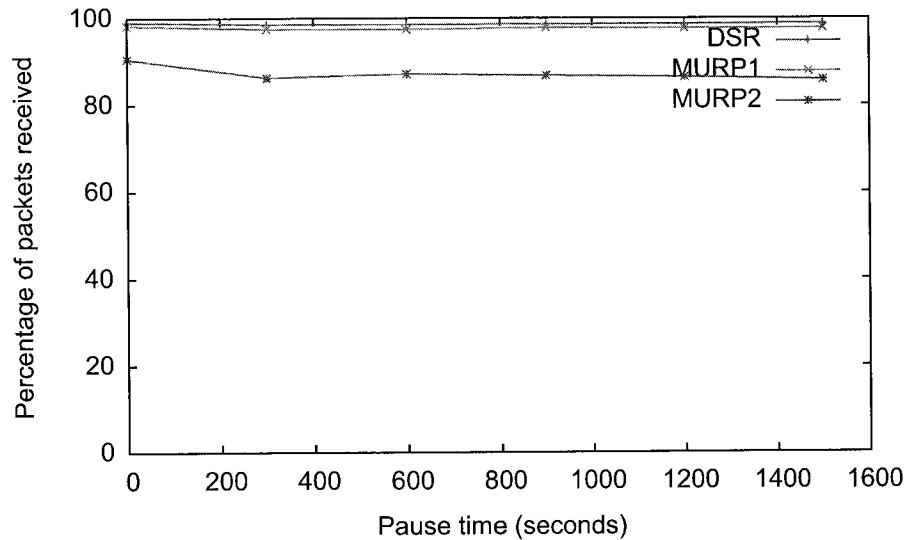


Figure 7-6: Simulation results: percentage of packets received for transmission ranges 2500 meters.

it is the intended destination. The Route Discovery occurs regardless of this change once a new packet transmission is attempted by the application protocol, but it was theorized that the forced Route Discovery would ensure better route correction and reduce either the total route overhead or the reliability of the protocol. However, the results for MURP2 show that, while it achieves a reduction in total overhead and still makes moderate gains in recovery speed under some conditions, a significant sacrifice in reliability is made.

The results displayed in Figures 7-1, 7-2 and 7-3 show that MURP has a consistent advantage over DSR in terms of recovery time, although the gap closes with increased pause time and for higher transmission ranges. This improvement is made at the sacrifice of approximately 2% of packet deliveries and an increase in number of transmissions, though each of these gaps is also reduced with increasing pause time and transmission range.

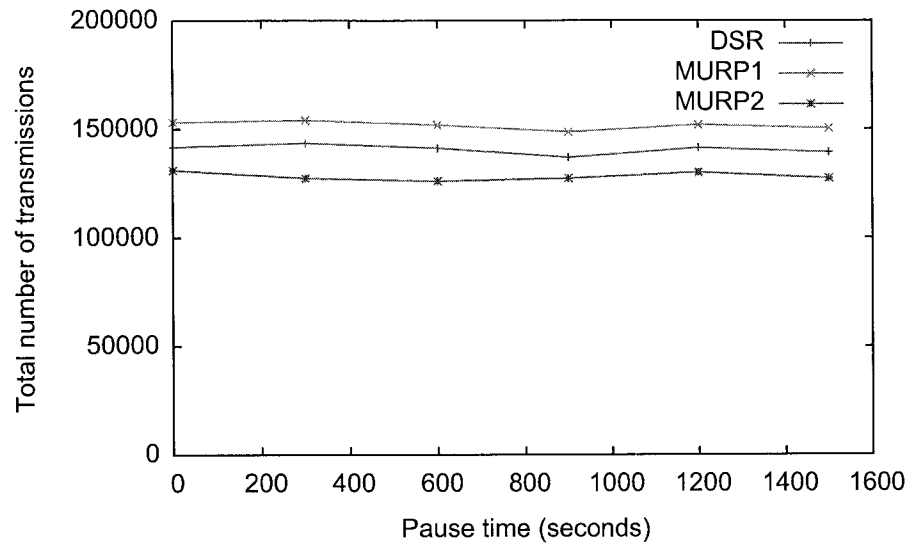


Figure 7-7: Simulation results: total number of packet transmission for transmission range 1500 meters.

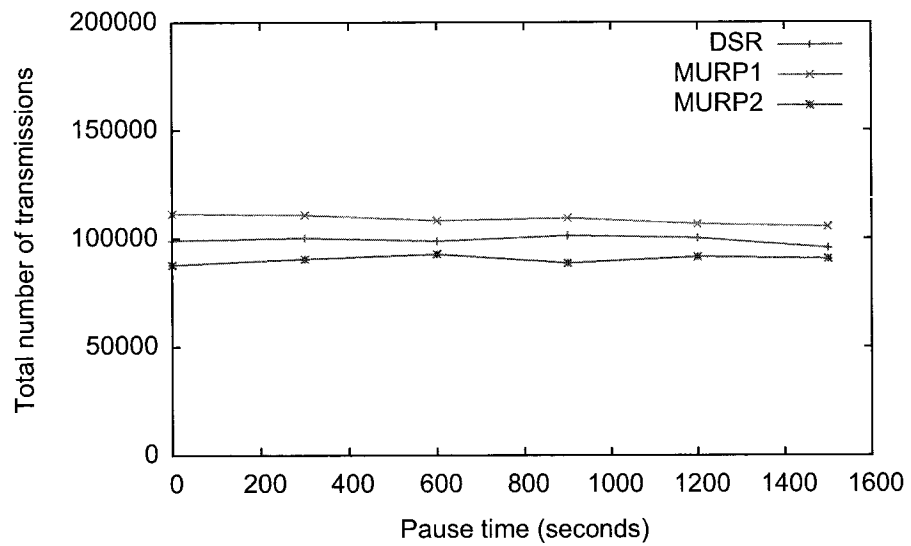


Figure 7-8: Simulation results: total number of packet transmission for transmission range 2000 meters.

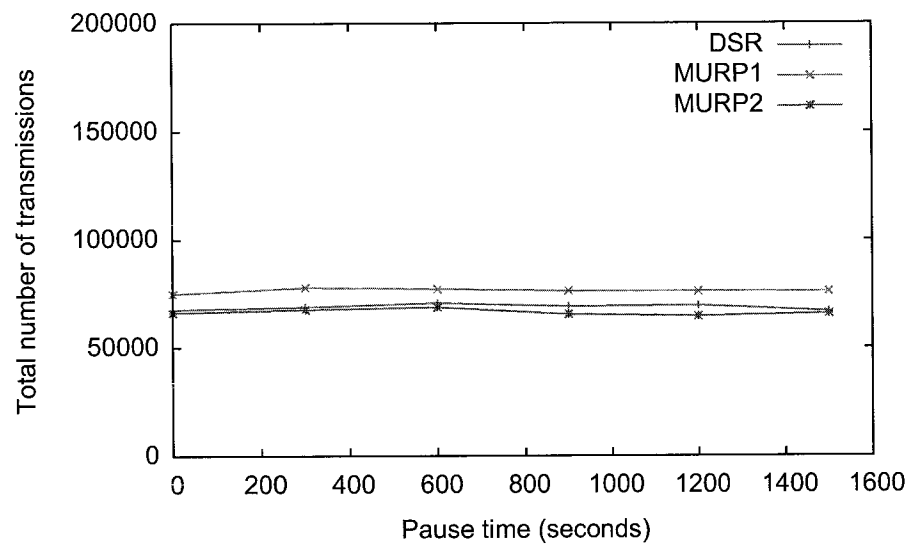


Figure 7-9: Simulation results: total number of packet transmission for transmission range 2500 meters.

## Chapter 8

# Conclusion

The original purpose of this study was to create a routing protocol which could meet the communication needs of an ad hoc network of autonomous undersea vehicles. These vehicles by definition operate in an environment which provides numerous obstacles to communication. Therefore the routing protocol designed for use with these vehicles has to be reactive and adaptable to frequent topology changes. Dynamic Source Routing is the most suitable starting point for such a study, as it is designed in a simple manner to avoid the unnecessary overhead which is associated with many other RF wireless routing protocols. We have designed the Mobile Undersea Routing Protocol which uses DSR as a framework and adds the Route Recovery mechanism to facilitate quicker and less expensive responses to errors.

We created a Java based network simulator which implements each of these protocols and provides a medium in which to test these and other protocols using variable topology and traffic settings. The simulator makes use of object oriented methodology and the Spring framework to allow for easy runtime adjustment of settings and replacement of protocols. We ran numerous tests and compiled data from the results using statistical analysis tools which are built into the simulator architecture. Additionally, a random waypoint mobility model and an underwater physical layer with random packet loss were implemented in order to test the protocols under error prone conditions. We judge the capability of each protocol based on time to recover from an error, number of packets used in communication and percent of data packets



which reached their destination. Each protocol is tested with varying degrees of interconnectivity, based on wireless communication range, and topology volatility, which is determined by pause time between mobile node movement. The data shows that MURP reacts more quickly to routing errors than DSR, particularly given a highly volatile topology. Because of the high latency of the underwater environment, and the independent nature of the AUVs, gaps in network availability are extremely costly. While a small percentage of reduced reliability and increased total packet transmissions was required to reach this goal, the drawbacks are outweighed by the benefits, particularly in rapidly changing topologies.

In future study, we plan to fine tune MURP to reduce packet loss while still maintaining the recovery time and efficiency of transmissions. We will create a more intricate method for determining the time to live used for each Route Recovery. We also plan to run more extensive testing using simulations with larger topologies and more traffic sources, such as 100-200 node topologies and topologies in which all nodes communicate with all other nodes. In doing so, we hope to develop configurations which allow MURP to be effective in any of these settings.

There are a number of features that we will implement in the simulator that were left out due to being unnecessary for this project, but which will be useful in other types of studies. One such feature is a graphical user interface and visualizer. We will implement the ability to easily reconfigure the protocols and topologies in a front end, run the simulation, and view the resulting network and activity as an Adobe Flash video. Java libraries are available for creating Flash video dynamically in code. The program will iteratively draw shapes for each node and packet at a chosen interval of time to provide the necessary granularity to portray an acceptable image of the motion of the nodes and the packets. The visualization program will be built as an additional protocol layer for which events can be generated by any user protocol

in a similar manner to which the Statistics class works. A transport layer and link layer will be added and implementations created for 802.11 as well as TCP and UDP protocols. In order to facilitate Media Access Control (MAC) layer research, a more correct implementation of the underwater physical layer will be created with more robust error models using the attenuation formula mentioned in Chapter 2. It will also correct for the problem of the intersection of the packet and the moving node for which we use a simplified formula in this project. More complicated traffic models will also be implemented, such as an exponential traffic distribution to demonstrate networks with more varying concentrations of traffic and congestion. These will also be useful in enabling the use of the simulator for MAC layer protocol testing. Since the simulator program makes use of object oriented principles extending it to enable the above functionality will involve minimal changes to the core simulator code.

Finally, we plan to implement more routing protocols, such as AODV, Waypoint Routing, and other protocols historically used for mobile wireless networks and underwater networks. Given the analysis of these protocols provided in Chapter 4, it would be interesting to understand the performance of these protocols in our simulated environment. It would also provide further verification to the simulator's functionality, as the performance of some of these protocols underwater is well documented. Most importantly, simulation results for these protocols will provide further comparisons by which to judge the effectiveness of MURP.

# Bibliography

- [1] R. Bai and M. Singhal. DOA: DSR over AODV routing for mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 5:1403–1416, Oct 2006.
- [2] L. M. Brekhovskikh and Y. P. Lysanov. *Fundamentals of Ocean Acoustics*. Springer, 3rd edition, Aug 2005.
- [3] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communication and Mobile Computing: Special Issue on Mobile Ad Hoc Networking Research Trends and Applications*, 2(5):483–502, 2002.
- [4] R. Castaneda, S. R. Das, and M. K. Marina. Query localization techniques for on-demand routing protocols in ad hoc networks. *Wireless Networks*, 8:137–151, 2002.
- [5] D. M. Crimmins, C. T. Patty, M. A. Beliard, J. Baker, J. C. Jalbert, R. J. Komerska, S. G. Chappell, and D. R. Blidberg. Long-endurance test results of the solar-powered AUV system. In *MTS/IEEE Oceans 2006*, Boston, MA, Sept. 2006.
- [6] J. C. Jalbert. Multiple AUV communications test report - Lake George, NY; October 17 - 22, 2004. Technical Report 0411-01, Autonomous Undersea Systems Institute, Nov. 2004.
- [7] J. C. Jalbert, J. Baker, J. Duchesney, P. Pietryka, W. Dalton, D. R. Blidberg, S. G. Chappell, R. Nitzel, and K. Holappa. Solar-powered autonomous underwa-

- ter vehicle development. In *Thirteenth International Symposium on Unmanned Untethered Submersible Technology (UUST'03)*, Durham, NH, Aug. 2003.
- [8] D. B. Johnson, D. A. Maltz, and J. Broch. DSR: The dynamic source routing protocol for multi-hop ad hoc networks. *Ad Hoc Networking*, pages 139–172, 2001.
- [9] D. B. Johnson, D. A. Maltz, and Y.-C. Hua. The dynamic source routing protocol for mobile ad hoc networks (DSR). RFC 4728 (Standard), 2007.
- [10] D. E. Lucani, M. Médard, and M. Stojanovic. Underwater acoustic networks: Channel models and network coding based lower bound to transmission power for multicast. *IEEE Journal on Selected Areas in Communications*, 26:1708–1719, Dec 2008.
- [11] C. E. Perkins and E. M. Royer. Ad-hoc On-Demand Distance Vector Routing. *MILCOM'97 panel on Ad Hoc Networks*, Nov. 1997.
- [12] J. G. Proakis, E. M. Sozer, J. A. Rice, and M. Stojanovic. Shallow water acoustic networks. *IEEE Communications Magazine*, 39:114–119, Nov 2001.
- [13] A. H. Quazi and W. L. Konrad. Underwater acoustic communications. *IEEE Communications Magazine*, 20:24–30, Mar. 1982.
- [14] J.-H. Song, V. W. Wong, and V. C. Leung. Efficient on-demand routing for mobile ad hoc wireless access networks. *IEEE Journal on Selected Areas in Communications*, 22:1374–1383, Sep 2004.
- [15] D. A. Tran. Congestive adaptive routing in mobile ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 17:1294–1305, Nov 2006.

- [16] L. Wang and Y. Xiao. A survey of energy-efficient scheduling mechanisms in sensor networks. *Mob. Netw. Appl.*, 11(5):723–740, 2006.
- [17] W. Ye, Heidemann, J., and D. Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 12(3):493 – 506, june 2004.