Design of a Routing Protocol for Enhancing Underwater Transmission

Mohini Yadav
Dronacharya College of Engineering

***Abstract*— 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 Acoustic 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 C++ 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.**

I. INTRODUCATON

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.

II. OBJECTIVE

We decided on two goals for our protocol (ARP):- **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 Acoustic Routing Protocol (ARP) 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.

III. EXISTING SYSTEM

In this article , we describes the simulator program which was written for this paper. 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 Model View Controller architecture. Event based processing is shown to be used through a simple Queuing architecture. The program is configurable using the XML configuration files to analyze the various topologies and communication parameters. Design decisions are explained with regards to the implementation of the OSI model for network communication.

Random Waypoint Simulator: we created network simulator program in the C++ programming language. This program is configurable using XML configuration file for parameters for simulator network and the topology for the AUVs. This allows us to modify parameters and the protocol to be used at runtime and get results for various topologies and the routing protocol of choice.

The second key component utilized in the simulator program is inheritance. While this is one of the most basic C++ 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 receives 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 performed within each protocol. 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.

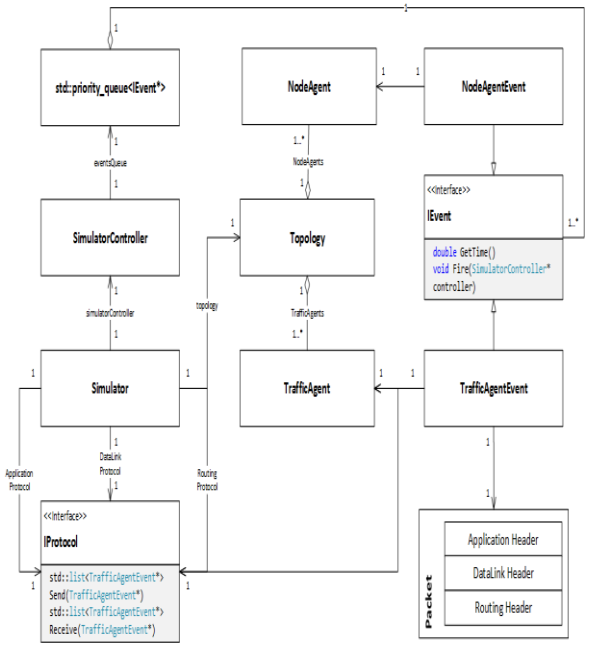

Fig.3.1 Simulator Class Diagram

Simulator Code Organization

```
// Logging for simulator

#pragma once

#include <iostream>
#include <iomanip>
#include <list>
#include "typedefs.h"
#include "Vector.h"

// Log the generic send and receive events when fired
void LogEvent(ID packetID, std::string mode, std::string protocol, ID agentID, Time eventTime);
// Log when the next agent on the route is outside the transmission range
void LogInRangeError(ID source, ID destination, double distance, double transmissionRange);
// Log when a packet is received at the destination agent's application layer
void LogPacketReceived(ID packetID, ID source, ID destination, std::list<ID> route);
// Log when a route requested has been received at the destination node
void LogRouteRequestReached(ID packetID, ID source, ID destination, std::list<ID> route);
```

```
// Log when a route request is broadcasted by a agent on the route
void LogBroadcast(ID packetID, int TTL, ID currentAgent, ID source, ID destination, std::list<ID> agentsInRange);
// Log when a route reply reaches the source and the route to destination is added to the routing table
void LogRouteAdded(ID packetID, ID currentAgent, ID destination, std::list<ID> routeToDestination);
// Log the node agent mobility events
void LogMobilityEvents(ID agentID, std::string mode, Time eventTime,
        Utilities::Vector from = Utilities::Vector(0, 0), Utilities::Vector to = Utilities::Vector(0, 0));

// Print the std::list<ID> as suited to the requirements
std::ostream& operator<<(std::ostream& stream, std::list<ID> list);
```

Event-queue architecture: An example of the ways in which events are handled in the queue is outlined in Figure 3.2 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.

Each protocol must generate the requisite events for the next layer in the stack. 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.

Upon initialization the simulator generates all the packets automatically for the specified total transmission time at the specified time interval, as well as the corresponding application layer send event, for each traffic agent and the events for initial movement of the node agents are also generated. The application layer protocol then executes the first 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. Averages and

standard deviations of each statistic item are output after all iterations have completed.

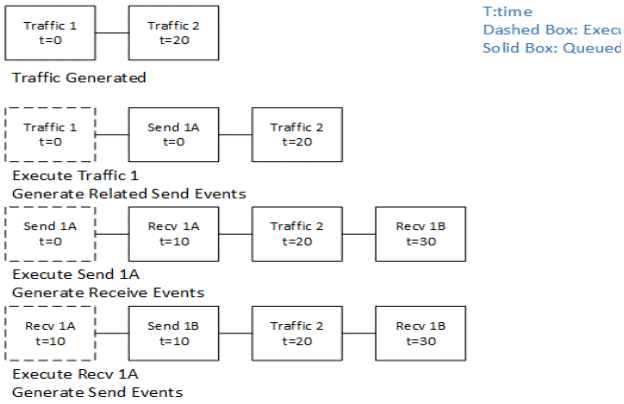

Fig-3.2 Event queue architecture example

IV. PROPOSED SYSTEM

In this article, we describe the proposed protocol, which adapts the principles of DSR for use in the underwater environment. The Acoustic Routing Protocol (ARP) 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.

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 Acoustic Routing Protocol (ARP) 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 example 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 of a Route Recovery are shown in Figure 4.1. This contrast sharply to using DSR's response to errors using Route Maintenance, 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 4.2, ARP'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 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.

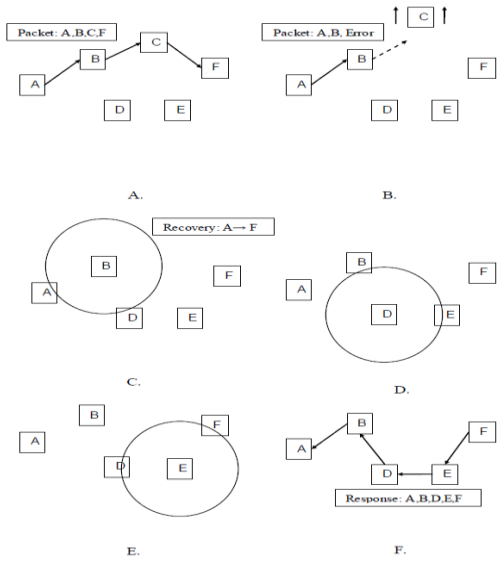

Fig. 4.1 an example of route repair

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 4.2, 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 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 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.

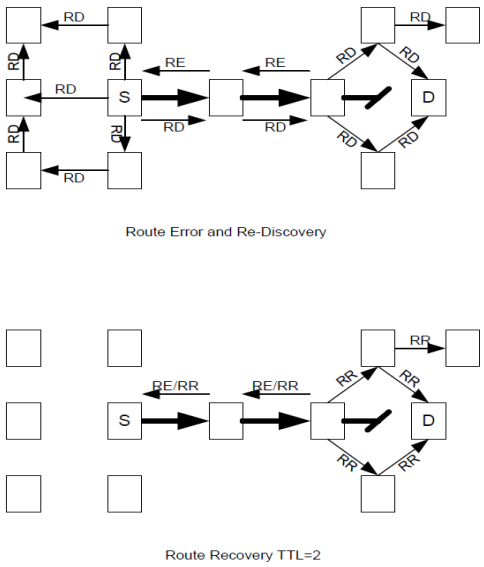

Fig. 4.2 comparison between route discovery and route recovery responses to error

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 ARP to be an efficient and robust protocol.

V. 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 Acoustic 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 C++ 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 lost 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 packet which reached their destination. Each protocol is tested with varying degrees of inter-connectivity, based on wireless communication range, and topology volatility, which is determined by pause time between mobile node movements. The data shows that ARP 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.

REFERENCES

[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 underwater 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. M_edard, 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.