INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS
ISSN 2320-7345

ACOUSTIC ROUTING PROTOCOL FOR ENHANCING UNDERWATER TRANSMISSION

[1]Kavita, [2]Sunita

[1]Department of computer science and engineering
Bhagat Phool Singh Mahila Vishwavidyalaya, Khanpur Kalan
Sonepat, India
Yadavkavita42@gmail.com
[2]Lecturer in Dept of computer science and engg.
Bhagat Phool Singh Mahila Vishwavidyalaya
Khanpur Kalan, India

***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 thesis 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. Statistics were calculated based on packets delivered, total transmissions, and time to recover from a route error as measurements of protocol effectiveness.**

Keywords- acoustic routing protocol, enhancing underwater transmission,

I. 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 complete end-to-end packet delivery ratio and total transmission 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 efficiencies 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*. *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.

Underwater networks

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 route of the frequency decibels per kilometer for electromagnetic waves. The absorption of acoustic waves is by contrast several orders of magnitudes lower. 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. There is also variation in sound speeds due to changes in temperature, salinity, and hydrostatic pressure.

II. DYNAMIC SOURCE ROUTING PROTOCOL (DSR):

The *Dynamic Source Routing* protocol (DSR) is a simple and efficient routing protocol designed specifically for use in multi-hop wireless ad hoc networks of mobile nodes. DSR allows the network to be completely self-organizing and self-configuring, without the need for any existing network infrastructure or administration. The protocol is composed of the two mechanisms of *Route Discovery* and *Route Maintenance*, which work together to allow nodes to discover and maintain *source routes* to arbitrary destinations in the ad hoc network. The use of source routing allows packet routing to be trivially loop-free, avoids the need for up-to-date routing information in the intermediate nodes through which packets are forwarded, and allows nodes forwarding or overhearing packets to cache the routing information in them for their own future use. All aspects of the protocol operate entirely *on-demand*, allowing the routing packet overhead of DSR to scale *automatically* to only that needed to react to changes in the routes currently in use.

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 decrease the time taken to recover from an error. 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.

III. SIMULATOR

In order to test the proposed routing protocol, we use the C++ 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 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 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**. 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.
Simulation results: 30000 second pause time and 1500 meter transmission range
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 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 and the ARP protocol described in this paper. Other protocols described in our research will be implemented as part of future work. The success of the ARP 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 ARP 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.
The chosen settings of *30000 seconds for pause time and 1500 meters for wireless transmission range* represent a significant impact of mobility. For these results, ARP improves upon recovery time, while maintaining approximately equivalent packet delivery at a cost in overhead of about 11000 packet transmissions.

We have to include several graphs to show how each measure is affected by the degree of mobility in the topology as measured by pause time. Each graph will show 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 will show 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, 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 ARP. The application or transport layer would be responsible for ensuring lost packets are retransmitted. The results for packet delivery are only intended to show the general reliability of the protocol under given conditions.

ARP 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.

IV. SIMULATOR CONFIGURATION

Code:

```
<?xml version="1.0" encoding="utf-8"?>
<!--The dimensions are in meters-->
```

```
<!--The speed is in meters per second-->
<!--The time is in seconds-->
<NetworkSimulator
  TotalSimulationTime = "500"
  TransmissionInterval = "50"
  PropagationSpeed = "1500">
  <Topology
    type="Random Waypoint Model"
    numAgents = "15"
    gridX = "5"
    gridY = "5"
    dimensionCellX = "1000"
    dimensionCellY = "1000">
    <NodeAgent
      speed = "2"
      pauseTime = "50">
    </NodeAgent>
    <TrafficAgent
      transmissionRange = "2000">
    </TrafficAgent>
  </Topology>
</NetworkSimulator>
```

Result:

V. DSR ROUTE REQUEST AND ROUTE REPLY

Code:

```
#include "stdafx.h"
#include "IProtocol.h"
#include "TrafficAgent.h"
#include "TrafficAgentEvent.h"
#include "Packet.h"
#include "Topology.h"
#include <list>

/*
The send function for the DSR routing protocol:
The algorithm for sending the request for route discovery and route maintainenance are as follows:
1) For a RouteRequest
        > If the current traffic agent is not already on the route
                > Add it to the route
                > Broadcast the Route Request to other traffic agents in range
                  (See the broadcast function to know how the route request is sent)
        > otherwise, ignore the packet received as this agent is already in the route which implies we went in a loop
2) For a RouteReply

3) For a RouteError

*/
std::list<TrafficAgentEvent*> DSRRoutingProtocol::Send(TrafficAgentEvent* incomingEvent)
{
        std::list<TrafficAgentEvent*> generatedEvents;

        Packet* pRoutingPacket = incomingEvent->GetPacket();
        TrafficAgent* pAgent =          incomingEvent->GetAgent();
        ID destination =                pRoutingPacket->GetDestination();
        ID source =                                     pRoutingPacket->GetSource();

        switch (pRoutingPacket->GetRoutingPacketType())
        {
        case RoutingHeader::RoutingPacketType::RouteRequest:
        {
                std::cout << "Event fired: Send, Route Request, Agent:" << pAgent->m_id << " at " <<
incomingEvent->GetTime() << std::endl;

                ID currentID = pAgent->m_id;
                bool added = pRoutingPacket->AddToRoute(currentID);
                if (added)
                {
                        generatedEvents = BroadcastRouteRequest(pAgent, pRoutingPacket);
                }
                else
                {
                        std::cout << "Adding the current ID: " << currentID << " to the route "
```

```
                                        << pRoutingPacket->GetRouteToDestination()
                                        << " for the destination " << destination << " failed." << std::endl;
                        }
                        break;
                }
                case RoutingHeader::RoutingPacketType::RouteReply:
                {
                        // handle the route reply here
                        // Add the route to the source and the destination to the routing table of the current agent here
                        std::list<TrafficAgentEvent*> payloadEvents;
                        std::list<ID> routeToDestination = pRoutingPacket->GetRouteFromCurrentToDestination();
                        std::list<ID> routeToSource = pRoutingPacket->GetRouteFromCurrentToSource();
                        if (pAgent->m_id != destination)
                        {
                                // add the route to the destination
                                pAgent->AddRoute(destination, routeToDestination);
                                payloadEvents = GenerateEventsForPayloadPackets(pAgent, destination);
                                std::cout << "Adding route to " << destination << " in routing table for " << pAgent-
>m_id << std::endl;
                                std::cout << "The route is " << routeToDestination;
                        }
                        if (pAgent->m_id != source)
                        {
                                // Add the route to the source
                                pAgent->AddRoute(source, routeToSource);
                                payloadEvents = GenerateEventsForPayloadPackets(pAgent, source);
                                std::cout << "Adding route to " << source << " in routing table for " << pAgent->m_id
<< std::endl;
                                std::cout << "The route is " << routeToSource;
                        }
                        pRoutingPacket->MoveNextOnRouteMarkers();
                        generatedEvents.insert(generatedEvents.begin(), payloadEvents.begin(), payloadEvents.end());

                        if (source == pAgent->m_id)
                        {
                                std::cout << "The Route Reply has reached the origin of the Route Request" << std::endl;
                        }
                        else
                        {
                                ID nextAgent = pRoutingPacket->GetNextOnRouteToSource();
                                pRoutingPacket->nextOnRoute(nextAgent);

                                // Send the packet to the agent next on the route to source
                                TrafficAgentEvent* pEvent = new TrafficAgentEvent(
                                        GetSimulatorGlobals().GetGlobalTime(),
                                        pAgent,
                                        TrafficAgentEvent::EventMode::Send,
                                        Protocol::Physical,
                                        pRoutingPacket);

                                generatedEvents.push_back(pEvent);

                        }
                        break;
                }
                case RoutingHeader::RoutingPacketType::RouteError:
```

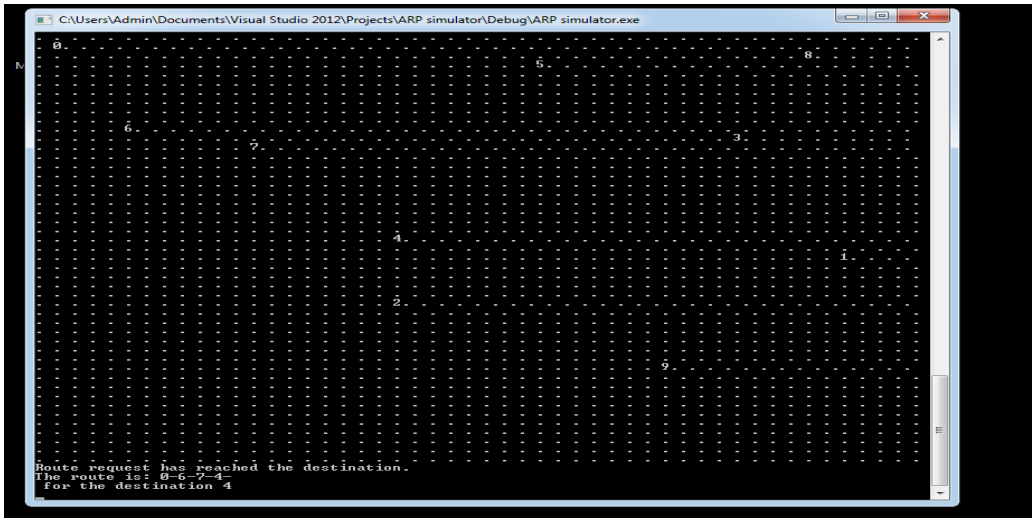
```
                break;
        }

        return generatedEvents;
};

/*
The list of actions that happen when a packet is received at the routing layer
1) For a Route Request
        > If the Route Request has reached the destination
                > If the packet is a duplicate Route Request (that is if a Route Request with a shorter path has already
                  reached the destination), ignore the packet
                > Else, create a Route Reply with the route till now and push the reply to the network
2) For a Route Reply
        > Generate a Send request for the same agent and the Reply will be handled there
3) For a Route Error

*/
std::list<TrafficAgentEvent*> DSRRoutingProtocol::Receive(TrafficAgentEvent* incomingEvent)
{
        std::list<TrafficAgentEvent*> generatedEvents;

        Packet* pRoutingPacket = incomingEvent->GetPacket();
        TrafficAgent* pAgent = incomingEvent->GetAgent();
        ID destination = pRoutingPacket->GetDestination();

        switch (pRoutingPacket->GetRoutingPacketType())
        {
        case RoutingHeader::RoutingPacketType::RouteRequest:
        {
                std::cout << "Event fired: Recieve, Route Request, Agent:" << pAgent->m_id << " at " << incomingEvent->GetTime() << std::endl; ID currentID = pAgent->m_id;
                if (currentID == destination)
                {
                        DepictGraphically(std::cout, *GetSimulator().GetTopology());
                        std::cout << "Route request has reached the destination." << std::endl;

                        // Add the destination to the route to complete it from source to destination
                        pRoutingPacket->AddToRoute(pAgent->m_id);

                        std::cout << "The route is: " << pRoutingPacket->GetRouteToDestination()
                                << " for the destination " << destination << std::endl;

                        // check for duplicacy using the packet's unique identifier
                        UniqueIdentifier packetID = pRoutingPacket->GetUniqueIdentifier();
                        bool isDuplicate = pAgent->HasAlreadyBroadcasted(packetID);

                        if (isDuplicate)
                        {
                                std::cout << "Route ignored" << std::endl;
                        }
                        else
                        {
                                ID source = pRoutingPacket->GetSource();
                                ID destination = pRoutingPacket->GetDestination();
```

```
                                std::list<ID> routeToSource = pRoutingPacket->GetRouteToSource();
                                std::list<ID> routeToDestination = pRoutingPacket->GetRouteToDestination();

                                unsigned int routeLength = routeToSource.size();
                                Packet* pReplyPacket = new Packet(
                                        Packet::Type::Routing,
                                        source,
                                        destination,
                                        routeLength, /* Time To Live */
                                        RoutingHeader::RoutingPacketType::RouteReply);
                                // Add the route to and from the source to the reply packet
                                pReplyPacket->AddRouteToSource(routeToSource);
                                pReplyPacket->AddRouteToDestination(routeToDestination);
                                pReplyPacket->ResetNextOnRouteMarkers();

                                // Add the unique identifier to the list of already broadcasted packets
                                // to mark the other Route Requests arriving as duplicate
                                pAgent->AddToBroadcasted(packetID);

                                // Generate a send event for the Route Reply
                                TrafficAgentEvent* generatedEvent = new TrafficAgentEvent(
                                        GetSimulatorGlobals().GetGlobalTime(),
                                        pAgent,
                                        TrafficAgentEvent::EventMode::Send,
                                        Protocol::Routing,
                                        pReplyPacket);
                                generatedEvents.push_back(generatedEvent);
                        }
                }
                else
                {
                        TrafficAgentEvent* generatedEvent = new TrafficAgentEvent(
                                GetSimulatorGlobals().GetGlobalTime(),
                                pAgent,
                                TrafficAgentEvent::EventMode::Send,
                                Protocol::Routing,
                                pRoutingPacket);
                        generatedEvents.push_back(generatedEvent);

                        std::cout << "Event generated: Send, Route Request, Agent:" << pAgent->m_id << " at "
<< generatedEvent->GetTime() << std::endl;
                }
                break;
        }
        case RoutingHeader::RoutingPacketType::RouteReply:
        {
                TrafficAgentEvent* generatedEvent = new TrafficAgentEvent(
                        GetSimulatorGlobals().GetGlobalTime(),
                        pAgent,
                        TrafficAgentEvent::EventMode::Send,
                        Protocol::Routing,
                        pRoutingPacket);
                generatedEvents.push_back(generatedEvent);
                break;
        }
```

```
		case RoutingHeader::RoutingPacketType::RouteError:
			break;
		}

		return generatedEvents;
}

/*
> If the current traffic agent has already braodcasted the packet (each packet has a unique identifier)
		> Ignore it
> else,
		> decrement the Time to Live and if the packet has expired ignore it
		> else,
			> Get the agents in the range
			> Create a new Route Request packet
			> Add the current traffic agent ID to the route
			> Send the packet to all agents in range
			> Mark the packet to be already broadcasted by this traffic agent.
*/
std::list<TrafficAgentEvent*>	DSRRoutingProtocol::BroadcastRouteRequest(TrafficAgent*	pAgent,	Packet*
pRoutingPacket)
{
		std::list<TrafficAgentEvent*> generatedEvents;

		if (pAgent->HasAlreadyBroadcasted(pRoutingPacket->GetUniqueIdentifier()))
		{
			std::cout << "The Route request has already been broadcasted from "
				<< pAgent->m_id << std::endl;
		}
		else
		{
			pRoutingPacket->DecrementTTL();
			std::cout << "TTL: " << pRoutingPacket->GetTTL() << std::endl;
			if (0 == pRoutingPacket->GetTTL())
			{
				std::cout << "ROUTING PACKET EXPIRED" << std::endl;
				std::cout << "The route is: " << pRoutingPacket->GetRouteToDestination();
			}
			else
			{
				ID currentID = pAgent->m_id;

				std::list<ID> IDs(GetSimulator().GetTopology()->AgentsInRange(currentID));
				std::cout << "Broadcasting the routing request to " << IDs;

				for (auto id : IDs)
				{
					Packet* pNewPacket = new Packet(*pRoutingPacket);
					pNewPacket->nextOnRoute(id);

					TrafficAgentEvent* pEvent = new TrafficAgentEvent(
						GetSimulatorGlobals().GetGlobalTime(),
						pAgent,
						TrafficAgentEvent::EventMode::Send,
						Protocol::Physical,
						pNewPacket);
```

```
                                generatedEvents.push_back(pEvent);
                        }

                        pAgent->AddToBroadcasted(pRoutingPacket->GetUniqueIdentifier());
                }
        }

        delete pRoutingPacket;
        return generatedEvents;
}


// If any packets are enqueued to be sent to a particular destination
// Add the events to send the packets again as the route has been found
std::list<TrafficAgentEvent*> DSRRoutingProtocol::GenerateEventsForPayloadPackets(TrafficAgent* pAgent, ID
destination)
{
        std::list<TrafficAgentEvent*> payloadEvents;

        std::list<Packet*> enqueuedPackets = pAgent->GetEnqueuedPackets(destination);
        for (auto pPacket : enqueuedPackets)
        {
                TrafficAgentEvent* pPayloadEvent = new TrafficAgentEvent(
                        GetSimulatorGlobals().GetGlobalTime(),
                        pAgent,
                        TrafficAgentEvent::EventMode::Send,
                        Protocol::DataLink,
                        pPacket);

                payloadEvents.push_back(pPayloadEvent);
        }

        return payloadEvents;
}
```

Result:



VI. FUTURE WORK

In future study, we plan to fine tune ARP 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 use 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 ARP to be effective in any of these settings.
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. 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 ARP.

VII. 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 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 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 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 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.

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. 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. 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.

VIII. REFRENCES

[1] *IEEE Transactions on Mobile Computing,*
[2] *Fundamentals of Ocean Acoustics. Springer, 3rd edition, Aug 2005.*
[3] *Wireless Communication and Mobile Computing: Special Issue on Mobile Ad Hoc Networking Research Trends and Applications,*
[4] *J. C. Jalbert. Multiple AUV communications test report - Lake George, October 17 - 22, 2004. Technical Report 0411-01, Autonomous Undersea Systems Institute, Nov. 2004.*
[5] *D. B. Johnson, D. A. Maltz, and J. Broch. DSR: The dynamic source routing protocol for multi-hop ad hoc networks. Ad Hoc Networking*
[6] *D. B. Johnson, D. A. Maltz, and Y.-C. Hua. The dynamic source routing protocol for mobile ad hoc networks (DSR).*
[7] *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,*
[8] *C. E. Perkins and E. M. Royer. Ad-hoc On-Demand Distance Vector Routing panel on Ad Hoc Networks, Nov. 1997.*
[9] *J. G. Proakis, E. M. Sozer, J. A. Rice, and M. Stojanovic. Shallow water acoustic networks. IEEE Communications Magazine,*
[10] *H. Quazi and W. L. Konrad. Underwater acoustic communications. IEEE Communications Magazine.*