Abstract—This report introduces various different ad-hoc routing algorithms and the high fidelity Urban Search and Rescue Simulator, before discussing the details for an implementation of the Ad-hoc On-demand Distance Vector routing protocol for USARSim.

I. INTRODUCTION

There has been a significant increase in the popularity and usage of mobile ad-hoc networks (MANETs) and multi-agent systems where it is not feasible to create a static network, due to the mobility of the member nodes. In such networks, due to the absence of dedicated routers, each member node is also responsible for routing messages to other nodes. Recently, there have been a number of ad-hoc routing protocols developed for MANETS, each with benefits relating to specific usage scenarios.

With the research focus in mobile robotics gradually shifting to multi-agent systems, and with the performance of such systems in many domains benefitting from reliable communication between the members of the team it is logical to take a look at the performance of the different ad-hoc routing algorithms in this field.

The Urban Search and Rescue Simulator (USARSim) [8] [9] is a newly developed, high fidelity mobile robotics simulator. Developed on top of the famous Unreal Engine 2.0, it uses the Karma Physics Engine to reliably and accurately model the physics behind the robots. It also provides a Wireless Simulation Server (WSS) [13] which that uses an approximation of the path loss between two robots to determine if a connection between them is possible.

This report documents the implementation of the Ad-hoc On-demand Distance Vector routing protocol for agents to be tested using USARSim. The existing structure of agents developed by the Jacobs University Robotics Group as described in [12] was extended to implement a simplified version of the AODV protocol.

This report shall first briefly introduce various ad-hoc routing protocols, before taking a look at the architecture of USARSim. Then the mechanism and interface of the WSS shall be introduced. Finally the existing structure and implementation of the routing protocol shall be explained, followed by a look at some results achieved.

II. AD-HOC ROUTING PROTOCOLS

Ad-hoc routing protocols can be classified primarily into two categories based on the method of exchanging state information:

1) Proactive protocols
2) Reactive protocols

While there exist other classifications such as location aided routing (Location Aided Knowledge Extraction Routing), quality of service supported protocols (Associativity-Based Routing), hierarchical protocols (Dynamic Address Routing) and power aware protocols (Energy Aware Dynamic Source Routing Protocol), these classifications are based primarily on the utility used to select a path.

A. Proactive protocols

Proactive or table driven routing protocols try maintain up to date routing information about the other nodes in the network. This is done by maintaining one or more routing tables, and exchanging routing information periodically (or when a routing entry is added or modified) to ensure that the routing tables are up to date and consistent. While the routing information for MANETs consisting of a small number of nodes does not cause a significant amount of overhead (while transmitting and storing), the performance does not scale well. In larger networks the overhead of maintaining fresh routing information might be too high for these protocols to be efficient. Also a change in the network topology often causes the routing to become unstable till the change is propagated through the network.

The Destination Sequenced Distance Vector (DSDV) routing protocol [2] is one of the earliest available protocols for ad-hoc networks. It is based on the Bellman-Ford algorithm, and it solves the routing loop problem by using destination generated sequence numbers. The Wireless Routing Protocol (WRP) [11] is another protocol based on distance vector routing. It stores the previous and next nodes to solve the routing loop problem. The Clusterhead Gateway Switch Routing (CGSR) protocol [1] is similar to the Border Gateway Protocol, in the sense that member nodes are grouped as clusters, with one node as a clusterhead. Gateways are those nodes that are members of two or more clusters. Routing algorithms such as DSDV are used to
maintain routing between clusters and, if needed, to maintain routing within a cluster.

B. Reactive protocols

Reactive or on demand routing protocols create routes only when required by a node. If a node requires a path to a destination, it starts a route discovery process in the network. In most such protocols, only the nodes along the path need to maintain information concerning the path. Such protocols often perform better than proactive protocols when implemented in a large networks due to a smaller overhead. However a large amount of network traffic can cause the performance to deteriorate sharply as most such protocols flood the network while looking for a route, and this can lead to clogging of links. Another major disadvantage is the delay required to find a route (as route discovery is initiated only when needed), which in some applications might be unacceptable.

The most common reactive routing protocol is the Ad-hoc On-demand Distance Vector (AODV) routing protocol [4]. When required, a node broadcasts a route request. If received by the destination or a node that has an active path to the destination, a route reply is sent back, otherwise the request is re-broadcast. There are many protocols like the Ad-hoc On-demand Distance Vector with Received Signal Strength (AODV-RSS) [7] that build on the AODV protocol. AODV-RSS places a constraint on the usable links over which a request is sent by predicting the time required to transmit the message to the destination and using the rate of change in the received signal strength to predict the link available time. It requires that the link available time is more than the time required to transmit, so that a new route would not have to be found midway through a message. The Dynamic Source Routing (DSR) protocol [5] is similar to AODV, except that the entire path is stored at the source, and is included in each packet sent along the path. This increases the robustness as the routing is not dependent on the routing tables at intermediate nodes, but also adds an overhead by increasing the message header size.

As discussed above, the advantages and disadvantages of each class of protocols is dependent not only on the size of the network, but also on the nature and amount of traffic generated. Therefore the performance of ad-hoc routing protocols cannot be accurately estimated without considering the application domain, and the system behavior (with respect to multi-agent systems).

III. URBAN SEARCH AND RESCUE SIMULATOR

As mentioned above, the Urban Search and Rescue Simulator is built on top of the Unreal Engine, and uses the Unreal Engine to simulate the environment and the robots.

At the start of the simulation a specified environment is loaded by the Unreal Engine. USARSim, via the BotServer and Wireless Simulation Server, provides an interface to spawn and control nodes, gather sensor information for various simulated sensors, and communication between the nodes.

A. BotServer

Adapted from the Gamebots project, the BotServer provides a network interface for users to spawn nodes, and then control their movement. Information gathered via various simulated sensors attached to the nodes can be retrieved over the network. Simply put, the BotServer provides a way to manipulate objects within the Unreal Engine.

B. Wireless Simulation Server

The Wireless Simulation Server uses an approximation of the path loss (the decrease of signal strength) from source to receiver to decide if a connection between to nodes is allowed. First described in [13], it has now been rewritten and the interface has been changed, but the working remains largely similar.

Started by USARSim when a simulation is started, it accepts connections over which various commands, such as registering, listening, opening a connection and getting the path loss between two nodes, are sent. Registering is required to map identifiers to network addresses, as using identifiers allows multiple nodes to be simulated on a single
computer with minimal work. On receiving a command to open a connection it checks if the specified node is listening, and if the path loss is better than a permissible value (-93 dBm) it connects to the specified robots at the specified ports, establishing a simulated connection.

Once the connection has been opened, the WSS allows sending of messages and closing the connection. Each time a message is sent, the path loss between the end points is checked, and if it is better than the threshold then the message is forwarded, otherwise the simulated connection is closed (the connection to each robot is closed. As the WSS is based on TCP, it only supports messages to be sent on an existing (or newly opened) connection.

While the WSS improves the validity of simulations using USARSim by imposing a restriction on inter node communication, it is not highly sophisticated from the point of view of a network simulator. The reason for this is that the behavior of the WSS is binary in essence. It either allows or disallows a connection, not reducing the bandwidth of the connection proportional to the path loss as occurs in reality. Furthermore, it does not have a bandwidth restriction for a connection, limiting it only by the processing speed of the server and the local network connections. Also, it does not consider physical interference or regulate access to the medium, two major limiting factors in wireless communication.

1) Approximating the Path Loss: The WSS uses RADAR propagation model suggested in [6] as a propagation model to approximate the path loss between the source and the receiver. It takes into account the distance between them, as well as the number of obstacles that lie in the path. According to the propagation model the path loss between two nodes is:

\[ P(d_0)[dBm] - 10n \log\left(\frac{d}{d_0}\right) = \begin{cases} nW * WAF & nW < C \\ C * WAF & nW \geq C \end{cases} \]

where:
- \( d \) Distance between the nodes in meters
- \( d_0 \) Reference distance in meters
- \( P(d_0) \) Path loss in dBm at \( d_0 \) metres
- \( n \) Attenuation factor with respect to distance
- \( nW \) Number of obstacles in the path
- \( C \) Maximum number of obstacles affecting the path loss
- \( WAF \) The attenuation factor for each obstacle

Earlier results [13] [10] show that the propagation model is highly accurate in approximating path loss values. However it requires a number of environment dependent parameters (above) to be ascertained for accurate results.

As the WSS is started by the USARSim server, it has access to the simulation information, more particularly to the positions of the nodes. Apart from using this to get the distance between the nodes, it recursively uses the ray trace functionality provided in Unreal Engine to count the number of obstacles between the nodes. To calculate the path loss the distance and number of obstacles are used in the above formula, along with parameters from a configuration file.

IV. THE SIMPLIFIED AD-HOC ON-DEMAND DISTANCE VECTOR ROUTING PROTOCOL

While a proactive routing protocol has been implemented [12] with reasonable results, the motion of the robots was restricted to try and maintain a routing structure once initialized. While this sufficed in the domain of autonomous exploration explored there, imposing such a restriction is very difficult in most other domains.

A simplified version of the Ad-hoc On-demand Distance Vector routing protocol [4] was implemented. AODV was chosen as it is one of the most common reactive protocols, and can be easily changed to one of the many AODV variants such as Ad-hoc On-demand Distance Vector with Received Signal Strength (AODV-RSS). It also has the benefit of using destination sequence numbers to avoid Counting to Infinity problem.

The simplified protocol will be described, closely following the AODV specification [4], and noting the simplifications made from the original protocol. After that the implementation will be briefly described.

A. The protocol

1) Routing Table Entry: A routing table entry contains the destination identifier, destination sequence number, flag indicating if the destination sequence number is valid, flags indicating the state of the entry, the next hop along the route, number of hops to the destination, life time and list of precursors. A precursor is a node that may use the route (and hence need to be notified in case of route breakage).

2) Messages: Route Request, Route Reply and Route Error messages are used.

A Route Request (RREQ) message contains the message type, originator and destination identifiers, the request id (a serial number, used together with the originator id to uniquely identify the request), the latest destination sequence number for the destination, the sequence number for the originator, the hop count (number of hops from the originator to the current node), a gratuitous route reply flag (indicating that a route reply is sent to the destination as well), the destination only flag (indicating that only the destination can respond to the request) and flag indicating if the destination sequence number is unknown. Multicast flags specified in the AODV protocol are not included.
A Route Reply (RREP) message is the response to a RREQ. It contains the message type, the destination and request originator id’s, the destination sequence number, the hop count from the originator to destination, and the lifetime of the connection. The multicast flags are again ignored. Also, an acknowledgement required flag, meant to be used over unreliable and unidirectional links to force route acknowledgements, are ignored, as is the prefix size field meant to allow use of routing prefixes.

A Route Error (RERR) message is sent when a broken link is detected. It contains the message type, number of unreachable destinations (caused by the broken link, or a RERR message), and unreachable destination identifier and sequence number pairs. The no delete flag is ignored, as we are ignoring the possibility of local repair of routes.

Additionally, as we discount unreliable and unidirectional links, the Route Reply Acknowledgement message has been ignored.

3) Destination Sequence Numbers: Destination sequence numbers must be updated in the routing table whenever a newer sequence number is received via a message. Also when the link to the next hop is broken, the corresponding destination sequence numbers must be incremented to ensure that only valid information is used to update the routing table.

Apart from updating the sequence number for other nodes, a node must also increment its sequence number immediately prior to generating a RREQ message. Also before a destination node replies to a RREQ, it must update its sequence number to the maximum of the sequence number it knows and the destination sequence number in the RREQ.

Care must be taken while comparing sequence numbers to take roll over into consideration.

4) Updating the Routing Table: A routing table entry is updated if the destination sequence number in the respective message is higher than the known sequence number for the destination, the two sequence numbers are the same but the new hop count is lower or if the sequence number is unknown. The lifetime is updated whenever a message is forwarded along the route, and the precursor list is updated if necessary.

5) Generating Route Requests: A RREQ message is generated when a node needs a route to a destination to which it does not have a valid path. The incremented sequence number and request id are used, along with the latest destination sequence number from the routing table. The hop count is set to 0, and other flags are set as needed.

The AODV specifications regarding controlling the rate of generation of RREQ’s is ignored for now.

6) Processing Route Requests: On receiving a RREQ, the path to the last hop is updated (or added with an unknown sequence number if needed). Then the validity of the request is checked by comparing the originator id and request id to previously received requests. Invalid requests are discarded.

For valid requests, the hop count is incremented. A reverse route to the originator is created or updated, including updating the destination sequence number for the originator if needed, setting the valid sequence number to true, setting the next hop to the originator as the node from which the RREQ was received, updating the hop count from the RREQ and the lifetime.

If a RREP is not to be generated, the message is updated (the hop count is incremented and the destination sequence number is updated if needed). The message is then broadcast. Notably, the node must not update the destination sequence number from the message.

If a RREP is to be generated (discussed later), the RREQ is discarded.

7) Generating Route Replies: A node generates a Route Reply to a RREQ if it is the destination or if it has a valid route to the destination with a better (or same) destination sequence number and the RREQ has the destination only flag not set. The RREP is sent to the originator using the reverse route set up during transmission of the RREQ.

If the node generating the RREP is the destination itself, it increments its sequence number if the sequence number in the RREQ is equal to the incremented value. It copies the destination and originator id’s from the RREQ and sets the destination sequence number to its sequence number, the lifetime to a specified value and the hop count to 0.

If the node is an intermediate node, it updates the route entry for the destination by adding the last hop of the RREQ to the precursor list. Also, the next hop towards the destination is added to the precursor list for the reverse route to the originator (of the RREQ). The hop count to the destination is set as the hop count in the RREP, the appropriate destination sequence number is copied from the routing table and the lifetime is set to the time of the route to the destination.

Gratuitous RREP messages are generated if the appropriate flag is set in the RREQ. A RREP containing the RREQ originators id as destination id, RREQ destination id as originator id, hop count to the RREQ originator as hop count and lifetime for the route to the RREQ originator as lifetime is sent to the destination from the RREQ.

8) Processing Route Replies: On receiving a RREP, if needed the node creates or updates a path to the previous
hop. The hop count is incremented, and if needed a route to the destination is created in the routing table. If a route to the destination already exists, it is updated if the sequence number in the routing table is invalid, the sequence number from the RREP is greater than the valid sequence number from the routing table, the sequence numbers are the same but the route in the routing table is inactive or if the sequence numbers are the same but the incremented hop count is smaller than the hop count in the routing table.

Creating or updating the route to the destination consists of marking the route as active, marking the destination sequence number as valid, setting the next hop to the previous hop (of the RREP), setting the hop count to the incremented hop count, setting the expiry time according to the lifetime in the RREP and setting the destination sequence number to that in the RREP.

If the node is not the originator from the RREP and the RREP has been used to update the routing table the RREP is forwarded towards the originator. If forwarded, the precursor list for the originator must be updated to include the previous hop of the RREP and also the next hop towards the originator has to be added to the precursor list for the destination from the RREP.

AODV Specifications about setting the Acknowledge flag in the RREP has been ignored.

9) Route Expiry Route Error Messages: A node starts processing a RERR message when it detects a link breakage while trying to transmit a message, when it receives a packet for a node for which it does not have an active path or when it receives a RERR about one or more routes. Processing involves invalidating the existing routes, creating a list of affected destinations, detecting which neighbors might be affected, and sending the appropriate RERR messages.

In the first case a list of unreachable destination consisting of the unreachable node and all nodes using the unreachable node as the next hop is created. In the second the only unreachable destination is the one the message was for. In the third case the list consists of those destinations mentioned in the RERR message for which the person sending the RERR is listed as the next hop.

The new RERR message contains the destinations from the list of affected destinations that have a non empty precursor list. The message is sent to those neighboring nodes that are on the precursor list of at least one destination in the message.

Before transmitting the RERR message, the routing entry for each affected destination must be updated as this might affect the destination sequence numbers in the message. For each affected destination, the destination sequence number is incremented (in the first two cases above) or copied from the incoming RERR message that caused the transmission. The routing entry to each of the affected destinations is invalidated, and the lifetime is set to a specified period after which the entry can be deleted.

Again specifications about limiting the number of RERR messages generated and prefix routing has been ignored. Also the possibility of broadcasting RERR messages instead of unicasting it to multiple neighbors has not been considered due to limitations in the server (discussed later).

B. Implementation architecture

The routing protocol was implemented extending the architecture described in [12].

The AODV_RREQ, AODV_RREP and AODV_RERR messages were defined. Using the existing classes and keeping the same interface for transmitting messages, the AODVModule was inherited from ComClass to perform the AODV operations, while other necessary changes were made to allow the agents to use the AODV protocol instead of the implemented protocol.

V. Results

Running the implemented AODV protocol, measurements were skewed by the performance of the USARSim server. The average throughput of the WSS was noted to be about 4 kbps (with 2 nodes spawned). This meant that in 10 minute simulation, there was significant lag by the end of the run. Also, increasing the number of nodes caused the server performance to deteriorate sharply. All these mean that currently no meaningful results could be achieved.

After a brief analysis of the working of simulation using UnrealProfiler and logs generated, a few ideas to improve the performance of the WSS were noted.

As performing ray traces is one of the most expensive operations performed by the WSS, caching the signal strength values will improve the performance (in our usage scenario). However this might not always be the case as it is dependant on the usage, since the frequency of updating the cached signal strengths might be more than if the ray tracing was done only when needed.

Another major bottleneck for the server is the fact that the UnrealEngine class to handle TCP connections allows only 255 bytes to be sent at a time (effectively a Maximum Transmission Unit of 255 bytes). Since this cannot be changed, we must hope that future releases of the UnrealEngine will allow a larger MTU.

Currently the WSS does not have any support for broadcasts. Performing a broadcast essentially means trying
to opening a connection to all other nodes, and sending the broadcast message when successful. Implementing some sort of broadcast functionality on the server might improve the performance of the server.

Another drawback is the fact that the WSS was designed as a simulator for connection based communication. This means that even if a single message needs to be transmitted (for example a RREQ message), a connection between the nodes must be set up. This makes using a routing protocol such as AODV quite inefficient. Adding support for UDP in the WSS should greatly help the performance.

VI. CONCLUSIONS

As can be seen with the previous section, a large number of improvements still need to be made to the wireless simulation in USARSim. Though this is currently disappointing with respect to testing out various routing protocols, it is understandable as USARSim is a relatively new simulator, and is still under development with new features being added all the time.

The next efforts must be directed towards improving the performance of the simulator, and then considering improving the fidelity (as mentioned previously the WSS still ignores a large number of factors in wireless communication).

In the meanwhile, more protocols should be implemented, so that when the simulator performance allows, or another appropriate simulator is developed, the performance of various protocols can be compared under different usage scenarios.

REFERENCES