PROJECT SNOW

LI CHARLIE(100832579)
THOMPSON KYLE (100936817)
ZHANG HECTOR(100947935)
APRIL 15, 2016

Table of Contents

List of Figure

1. Introduction

1.1 Context

The defining feature of an ad hoc network is its independence from the larger "internet". Ad hoc networks exist dynamically among a group of computers (often referred to in this report as nodes) communicating wirelessly with each other. It is also dynamic in the sense that new nodes can join and leave the network seamlessly without (in most cases) affecting the larger network and is completely decentralized from a main server.

1.2 Problem Statement

In this project we attempt to implement a routing protocol based on the Optimized Link State Routing (OLSR) protocol. Full compliance to the standard is not the goal but rather basic routing between intermediary nodes. Specifically, we aim to be able to transmit Transmission Control (TC) messages throughout the network and properly establish Multipoint Relays (MPRs) for the nodes.

1.3 Result

Ultimately we accomplished what we set out to achieve and then some. Exceeding the bare necessity for 3 node communication by implementing a fully compliant MPR computation algorithm and routing protocol along with correct hello and transmission control messages. The project has met and exceeded all goals set.

1.4 Outline

The rest of the project report is as follows. Section 2 discusses some of the background information and history on ad hoc networking and ad hoc network routing. Section 3 describes what we were able to achieve in the project. Section 4 evaluates the extent of which we were able to achieve our goals of intermediate node routing and dynamic network topology. Section 5 offers in conclusion a summary of the project and notes on future work.

2. Background

2.1 MANET

A mobile ad hoc network (MANET) protocol is a way in which packets can be routed through an ad hoc network from one node to another. These protocols can be either proactive, reactive, both or hierarchical. These types of protocols differ primarily in their knowledge of the network topology at any given time. Proactive routing protocols will send TC messages throughout the network at consistent intervals in order to maintain an accurate depiction of the network's topology. This has the advantage then of being able to more efficiently route direct communication packets from one host to another but has the downside of requiring the transmission control packets to circulate the network so frequently. On the flipside, reactive protocols do not rely on any state but instead flood the network for each direct message. This method has the advantage of not requiring TC packets to clutter the network but then also requires all messages flood the network to reach their destination. Hybrid and hierarchical protocols attempt to mix the benefits of reactive and proactive protocols into one with varying degrees of success. They are not so relevant to this project however.

2.2 OLSR

As the "Optimized" in Optimized Link State Routing might suggest, OLSR is a proactive ad hoc network protocol that attempts to optimize the number of messages required for one node to communicate with another. In contrast to traditional link state routing, OLSR accomplishes this by creating a subset of nodes called Multipoint Relays (MPRs) which form a kind of backbone for the ad hoc network. Whenever a host wants to send a message then it first sends it to one of its MPRs. The node will know which to send to based on an internal routing table that is built through TC messages. This MPR then forwards the message to another MPR (again, knowing which to send to based on a routing table.) until the destination is ultimately reached. The routing table mentioned is computed and maintained via the transmission control messages sent from all the hosts. It is the arrival of this message (or lack thereof) that informs a node about the existence of another node along with information on its of neighbors for routing.

2.2.1 OLSR Routing Protocol

- Initialize each node to bind to wireless interface, the MAC address is now your address
- Build a Hello message - used to notify the other nodes within reach of your presence
- Send Hello and Receive Hello Message
  - Update our node's state, which saves the neighbors we are connected to
  - The preceding hello messages will include the address of all our neighbors
- When receiving a hello message, the originator is our neighbor, which we have saved

- ❖ The message will also contain our neighbor's neighbors. Which we will call, our two hop neighbors, they get saved into our state because they become important later in MPR calculation
- ❖ An elaboration of this procedure is explained in 2.2.2

- ➢ Select our multipoint relays (MPR) nodes, which are essentially our neighbors that have access (or connection) with some of our two hop neighbors.
- ➢ After we select our MPRs, we build a topology control (TC) message with the addresses of our MPRs in it. This message gets flooded across the network and is forwarded only by nodes taking on the MPR role. This message lets other nodes know you are in the network and can be reached through a chain of forwards, starting at your MPR.
- ➢ Every node repeats this process and the network is established through a series of synchronized MPR notifications.

2.2.2 OLSR Routing Topology

This overview section will cover how a node is discovered by other nodes. When we plug in a wireless interface, a dongle we will call it, radio signals are emitted and received through the 2.4Ghz or 5Ghz band. In our project we turned our dongle's Ad-Hoc mode on through the linux command iwconfig so serve as the network interfaces of our network. In order to exchange information with other dongles, there's a specific setting that needs to be set in order to setup this connection.

Extended Service Set Identification (ESSID) is used to identify an access point, or node. Since all MANETs are typically access points configured in an Ad-Hoc fashion, we all need to use the same ESSID to connect to each other. Spread across our wireless frequency, we should all typically set up our dongle to all be listening to the same channel. There are 14 channels in WLAN, and picking one that has the least noise well help the performance of your network. The access point id is useful to set in the configuration too, it provides a address for the access point to connect to. When the settings of the network interface match across all nodes, then information is ready to be sent.

The information used to identify / discover nodes goes as follows.

- ➢ Send out a hello message to let others know you are here
- ➢ Receive a hello message sent over the wireless connection
  - ❖ Add the sender of this message to your one hop neighbor set. Anyone who is able to reach you through hello message is considered to be connected directly to you.
  - ❖ Inside the message that's broadcasted, has information on your neighbor's neighbors. These neighbor's neighbors are now your two hop neighbors. You can't directly send a message to them as they are too far, but you certainly know you can reach them through the sender of the hello message! In addition to knowing who your neighbors are, we can also determine if the edge between two nodes is bidirectional when both nodes can send hellos to each other.

- Selecting your MPR requires each node to calculate the number of two hop neighbors they can reach from this one of its one hop neighbors. The more you can get to, the more likely this neighbor will be considered as your MPR (when two neighbors are tied on the covered neighbors count, the degree of that neighbor is taken in account, more on that later).
- Now that you've selected some neighbors as your MPRs, the next step is to notify everyone. Time to send off some TC messages with the list of MPRs you selected.

- What happens when a neighbor leaves the network?
  - When we added the neighbor's address to our database for the first time, we also put a timer on his address. This timer is generally 6 seconds, and should give time for about 3 rounds of hellos to be sent by this neighbor before the timer times out. If the neighbor hasn't given you a hello in 6 seconds, you can delete him from your state and consider he is lost/disconnected.
  - A TC message will follow and notify the network of your new realizations of the surrounding neighbors (lost node).

2.2.3 OLSR Routing Message

This section will touch base on the types of packets we send in OLSR. They are a personal interpretation, based on RFC 3626, of what these fields mean. Some of the fields depicted in the messages not considered in our implementation of OLSR.

- General OLSR packet (width is 32 bits). All MAC addresses are actually 48 bits long, so extended the address space to 48 bits.

In the specification, an optimization to reduce network traffic was the ability to piggyback packets. This means one OLSR packet can contain multiple types of messages that were queued up over time before sending the a bulk messages off.

Packet Length		Packet Sequence Number
Message Type	Vtime	Message Size
Originator Address		
Time To Live	Hop Count	Message Sequence Number
…		
Message		
…		
Message Type	Vtime	Message Size
Originator Address		
Time To Live	Hop Count	Message Sequence Number
…		
Message		
…		

Figure 1 Packet Format

- Hello Message (The actual message)

The special attribute about hello message is the Link Code. Link code is a 8 bit integer that tells you two different things. The first 2 bits from the least significant tell you the kind of link that this is. By link I mean whether the connection is single directional, bi direction, if the link is actually lost, or an unspecified link which will likely result in the message being discarded. The next 2 bits define the type of neighbor the interface address is associated with. A node can have multiple interfaces and all of them are considered under the same link code. The types of neighbors we have in OLSR include bidirectional neighbor which is a neighbor that can be elected as MPR, a real MPR neighbor with message forwarding duties, and the default, not a neighbor class for arbitrary error cases.

<table>
<tr><td colspan="2">Reserved</td><td>Htime</td><td>Willingness</td></tr>
<tr><td>Link Code</td><td>Reserved</td><td colspan="2">Link Message Size</td></tr>
<tr><td colspan="4">Neighbor Interface Address</td></tr>
<tr><td colspan="4">Neighbor Interface Address</td></tr>
<tr><td colspan="4">…</td></tr>
<tr><td>Link Code</td><td>Reserved</td><td colspan="2">Link Message Size</td></tr>
<tr><td colspan="4">Neighbor Interface Address</td></tr>
<tr><td colspan="4">Neighbor Interface Address</td></tr>
<tr><td colspan="4">…</td></tr>
</table>

Figure 2 Hello Message Packet

- TC Message (The actual message)

These messages are special because of the advertised number sequence number (ANSN). They are a form of version control for the network. When a TC message is received, if the ANSN is lesser than the last TC message ANSN, it is then ignored. A topology update only occurs if the ANSN is greater than the ANSN value last seen. When a node detects a change in the topology, they increment this sequence number to show that a new version of the topology exists. The network tries to keep ANSN in sync, that's how it knows the knowledge of the topology is consistent throughout the network.

<table>
<tr><td>ANSN</td><td>Reserved</td></tr>
<tr><td colspan="2">Advertised Neighbor Main Address</td></tr>
<tr><td colspan="2">Advertised Neighbor Main Address</td></tr>
<tr><td colspan="2">…</td></tr>
</table>

Figure 3 TC Message Packet

3. Result

3.1 Design and Architecture

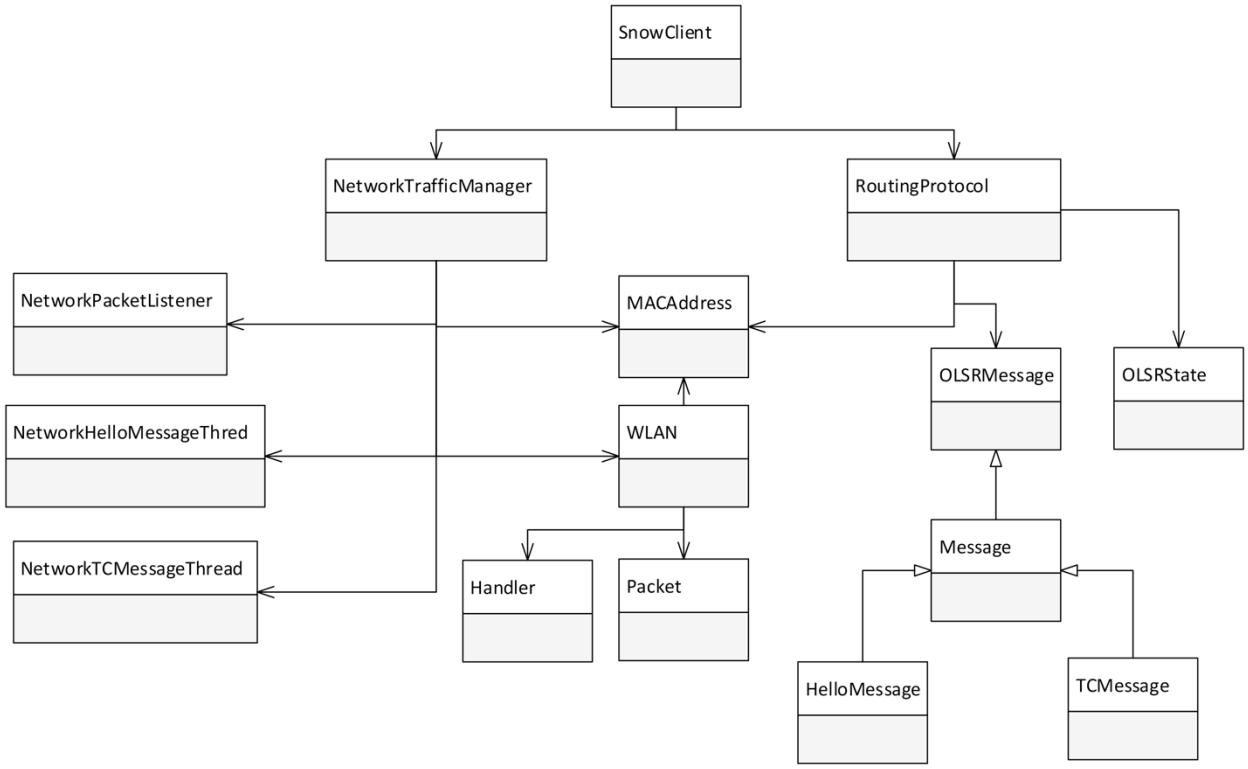

Figure 4 Snow UML

In Snow, there are three send threads which are used to broadcast the message to nearby nodes if they exist. Two of the threads are responsible for one of the two message types; hello or TC. The both the hello and TC messages are sent as a broadcast to anyone who can hear it. When a MPR designated node receives a TC messages, it will continue to flood this message towards the rest of the network. The third is a manager thread that only retrieves packets by our listening thread and sends out forwarded messages by the MPR.

We have a listening thread, one which picks up incoming hello and TC messages. This thread, along with our manager thread operating in a producer-consumer pattern. Listener is the producer of messages and manager is the buffer for the messages. There is a queue to save all received messages in, which is processed individually by the consumer, our lovely snow client.

When snow client, our controller class, retrieves the message from the queue, it passed the message on to the routing protocol. The OLSR routing algorithm takes it from there and will only return the message back to our controller if it requires forwarding. The following section is written to describe how we implemented this algorithm.

3.2 Libraries and Reference Material

In our project we made extensive use of the boost library to easily implement thread safe data structures and network communication. Specifically, from boost we used the concurrency library with threading and mutex for its ease of use. We also utilized the time class for our expiration times and timings. Originally the deadline timer from boost was used, but we opt it out in favor of handling our own timers by sleeping certain threads.

Our primary reference for the networking portion of the project was the open sourced NS-3 networking library. While we didn't use it directly, looking at the code was a great reference for how to build and manage the systems necessary for the OLSR protocol.

3.3 OLSR Topology

This section describes, in detail, the procedures we implemented to achieve each key OLSR activity. The algorithm is expressed in our personal words and interpretations gathered from RFC 3626 so it may be read and implemented by another person following our instructions.

3.3.1 Hello Messages

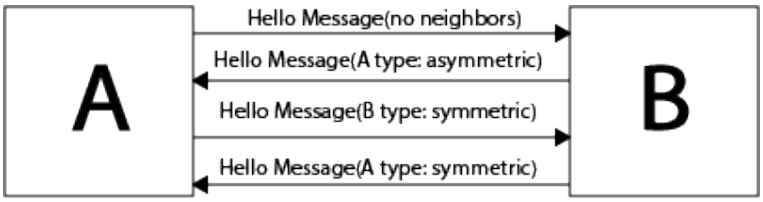

Figure 5 Hello Message - Simple Scenario

Hello messages serve to facilitate basic neighbor discovery. This is done by first broadcasting a hello message when a node A has no known neighbors. This hello message will continue to be broadcast until some other node B receives it and replies with it's own asymmetric hello message. Nodes A and B are now each other's neighbors and share a symmetric link between them.

3.3.2 TC Message

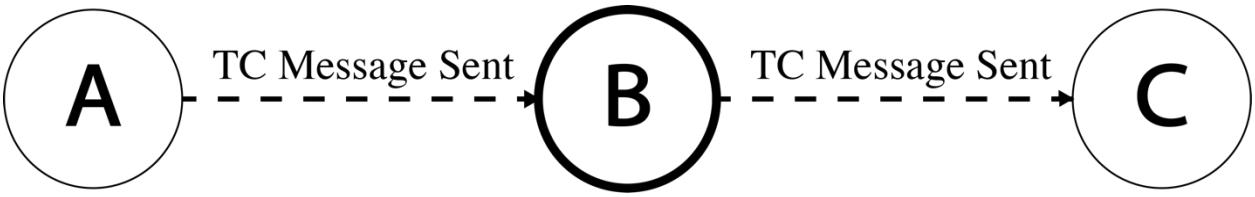

Figure 6 TC Message - Simple Scenario

TC messages serve as a form of "keep alive" message to be transmitted throughout the network, letting all other nodes know that a given node is still active in the network.

These messages also contain neighbor information that certain nodes may make use of for routing purposes. In the diagram above a TC message is shown being transmitted to B and then C. This then informs B and C that A is still active and can receive messages. Only nodes with the MPR status can forward TC messages, which is one of optimizations OLSR has done to avoid flooding duplicate messages across the network.

3.3.3 MPR

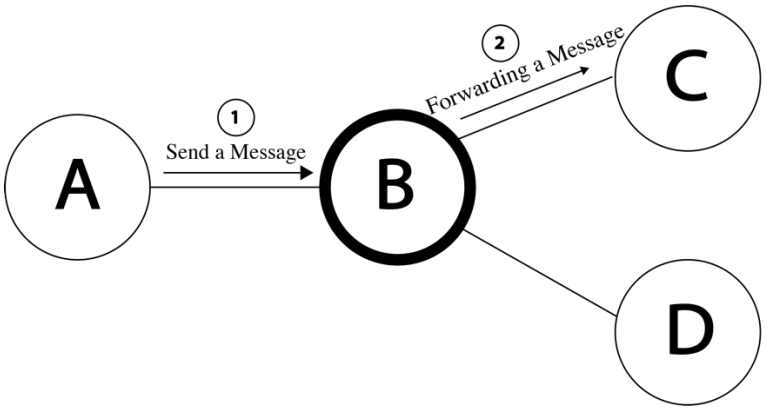

Figure 7 MPR - Simple Scenario

MPR nodes act as a backbone for the ad hoc network, serving as a form of router by directing packets from one source to another destination along its next hop in the network. In the example above the MPR node B is routing a TC message from A to C and D.

3.3.4 Join the Ad-Hoc Network – Route

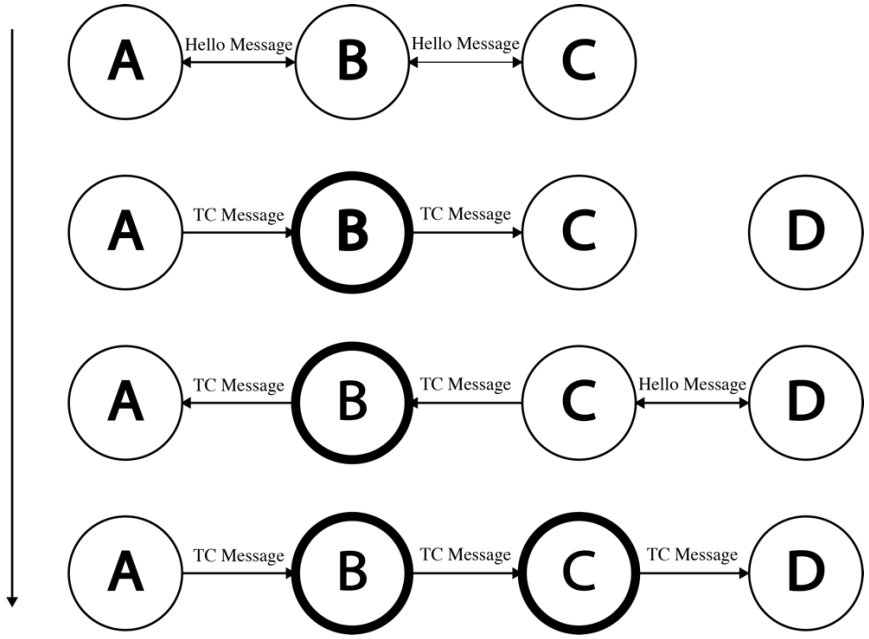

Figure 8 Route - Joining Node

Upon the initial broadcasting of your signal or a node in the network, you will have no neighbours. Your node will begin with no TC messages to send and your hello messages will have an empty neighbour list. The node will broadcast initial hello messages every two seconds. If another node is doing the same and is in range of your wireless receiver, you will receive a reply in the form of a hello message. If we receive a TC message before receiving any hello messages a priori, then the TC message is discarded since there is no record of neighbors on this node to select MPRs with. Upon receiving the first hello message it will then have the means of discovering the rest of the network through TC messages that are relayed to it via one or more MPR neighbors.

The first thing the node checks from the hello message is if the address of the sender is recorded as a LinkTuple inside the OLSR state. If it hasn't been seen the address before, then it creates a new record of this link and tags on an expiration time of six seconds. Currently the link is considered to be single directional. We won't know if the link is bidirectional until the neighbor has received our message, and included our address inside his hello message. In the case where we have gotten a message from the originator before, then we must already have a saved LinkTuple. We can then reset the timer for that LinkTuple we set initially to refresh the state of this neighbor by keeping it active. Appendix 8.5 and 8.5 has specific debug messages to show this state.

Each hello message will include a list of neighbor address of the sending node (our address can be in this list too!). We need to process each one of these addresses in order to find out who our two hop neighbors are. Once we extracted this information from the list, it's time to update our MPR states. Not all neighbors in this list are two hop neighbors, and we need to filter the address from this list that are already our one hop neighbor. To select our MPR correctly, we need to determine that we can reach our two hop neighbors from this neighbor that sent us the hello message. That is, we use our neighbor as the "bus" to our two hop neighbors. Generally, in OSLR implementation, there is the concept of willingness, which is the degree to which a node is willing to become the MPR or forward message on behalf of its neighbors. In our implementation, we set all nodes to have the maximum willingness (will always be MPR if possible) hence as long as there is a two hop neighbor that this neighbor can cover for us, this neighbor is likely to be chosen as MPR. An edge case we should consider here is if there were two neighbors with the same reachable two hop neighbor. In this case, we will choose the node with the highest degree (most amount of connected neighbors). Appendix 8.4 shows the print outs of our pre-defined MPR, becoming selected as candidate by its neighbors.

By this stage, we have a list of selected MPRs, we save all the possible MPRs that we have chosen into our database. On the next TC message to be send, we incorporate these selected MPR address inside out TC's Advertised Neighbors. From there on, my TC messages will be flooded into the network and the new node has officially joined the OLSR network. Appendix 8.4 shows the after an MPR has gotten a hello link message with his own address as MPR link type, this node knows that it's an MPR. Appendix 8.5 and 8.6 show that our MPR covers a two hop neighbor that can only be reached by him, therefore the MPR address is added to the list of MPR selectors.

Some good-to-know knowledge that we didn't implement is the act of updating the routing table. On each message receive, every node updates the routing table with the new knowledge they've received.

3.3.5 Loss of Connection - Re-Route

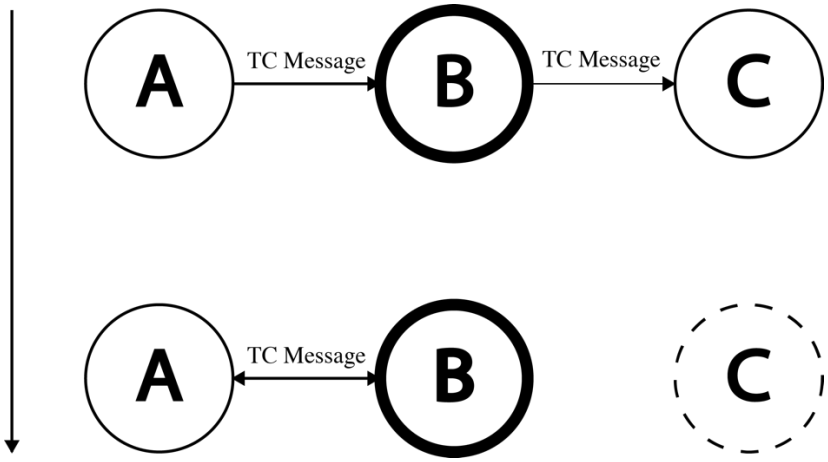

Figure 9 Re-Route - Neighbor Loss

When nodes leave the network their absence is made public by their missing hello messages in the network. As mentioned before, if a neighbor has not heard a hello from a node for six seconds, it will delete any records of this node and off it's databse. A throughout cleaning will begin, first wiping the LinkTuple, then it's known 2 hop neighbors, followed by its MPR selector. This big change will also trigger a new MPR calculation in case it was a MPR that dropped off the map. Following this expiry, the next time a node sends a TC message, these changes will appear inside its advertised neighbor selectors and the ANSN will be incremented, signalling a new version of the topology has been born. Rerouting occurs when MPRs are notified of this change and their routing tables updated correctly.

3.3.6 Detecting Nodes – Remove-Route

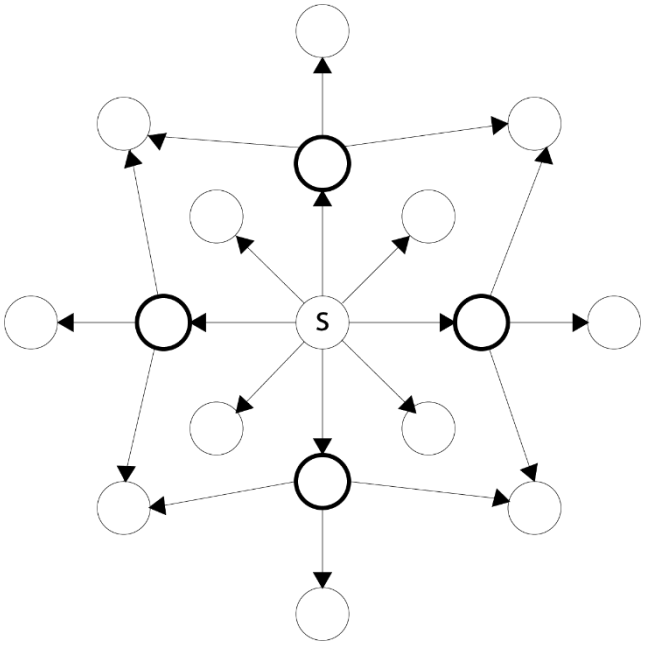

Figure 10 Remove-Route - Detecting Nodes

The only use for MPRs are for forwarding messages. The set of MPRs make up a spanning tree of the network. It is usually the most efficient way to get a message across the network. When a MPR receives a TC message for the first time, it remembers the sender's address (message originator) and the sequence number of this TC message. It's important to keep track these two parameters because TC messages are flooded to every MPR (non MPRs just read it and destroy it when they get it) and you don't want to process the same sequence number from the same originator twice. We can easily track something in a set of some sort. In the case where this TC message is unique, we can simply open up the message, increase the hop count by 1, and decrement the time to live by 1. We initially set a time to live to 255 hops, but this can be tuned based on your needs. Sometimes we will get a TC message where the number of advertised neighbors is zero. There is no point in flooding an empty message so we can silently discard this message. Since this will be the first time receiving this unique value, we can add it to our already seen TC messages database. If we always collect these messages, we will, at some point, fill up the entire space of the universe with old TC messages. So a scheduled timer for 30 seconds is started on each new TC message that removes this record on expiration. Appendix 8.1/2/3 show this TC traffic flow between the MPR and its neighbor nodes in a 3 node network. MPR forward all TC traffic, while edge nodes listen, and send their own TC periodically.

4. Evaluation

4.1 Node Scalability

Our testing environment embodied three to four nodes spread approximately 5 meters apart. In this configuration, hello messages are propagated to each and every node. In the canonical OLSR specification, this constructs the entire network into a one hop neighbor set. The proper way this can be resolved is to position our clients across a distance where the farthest nodes are not connected (packets not heard). Over testing, this distance became roughly 60 meters across open space, the entire length of the 1st floor extension of the Carleton library. That's going to be at least 90 meters between our farthest nodes! We couldn't develop in such an environment so we decided to blacklist our farthest nodes from each other, discarding all messages sent between them. This allowed us to successfully detect a two hop neighbor which was useful in MPR selection.

According to the behaviour described above and based on the set of specifications we implemented for this project (Evaluation 4.4), we believe our implementation will scale out to more than 3 nodes. We base our hypothesis on the importance of correct MPR selection and the roles they contribute to the expansion of our network. A recap of how this happens: a new node that join and will follow the HELLO protocol of OLSR, triggering a symmetric (bidirectional) link with some nodes in the network. Topology information is then shared between the neighbors of our new node. Each new node that joins the network triggers a refresh of our topology state through incrementing the ANSN value. The topology state database is recalculated through the current state of the node, incorporating the knowledge of neighbors and two hop neighbors. This process will repeat for each node that enters our network.

4.2 Performance Notes

We observed roughly 5 seconds of self configuration time to notify each node's presence within the network. This rough calculation is the sum of two hello intervals plus a random jitter time added. We decided to increase our intervals as opposed to decrease as defined in the RFC to slow down operations just as a design decision to help manage our debugging. Scaling upwards in node count, we expect to experience d * HELLO_INTERVAL time to fully configure, where d is the diameter of the network.

Our network, compared to real OLSR implementation, will generate more than a substantial amount of messages. We didn't utilize the full power of the generic message packet to encapsulate more than 1 actual message in each packet. That is, a packet should contain, multiple Hellos, TCs, and other OLSR packet types. This was rather an oversight on us, which will impact the scalability of our implementation. In our small test sandbox, this small oversight did not hinder the performance in any noticeable way. Distance between nodes was the biggest factor to consider for the latency of the messages.

Without blacklisting nodes to generate MPRs, we spread our clients across a distance of at least 90 meters, we noticed a slow down the messages received and an occasional dropping of the messages. This triggered a node disconnecting out of the network for a short amount of time until it's hello messages are received again. This experiment exposed a weakness in the protocol we implemented and allowed us to see link state protocols do not handle reliability of the links well. So long as the messages are synchronized across the network, a facade of well being is bestowed over the network.

4.3 Fault Tolerance

This subsection is actually given to us for free from the dynamic configuration of the protocol. In light of the well appreciated link state error tolerance properties of OLSR, we will focus this section on evaluating our implementation of the protocol and how it deals with our own technical glitches. Over the course of development, we've tried our best to deal with our own programming errors such as patching segmentation faults as they come up, but the one issue that occurs once in awhile is the issue of memory corruption. This issue is still at large during the writing of this report and detriments the reliability of a node. However, we think it provides a good example of the fault tolerances of the OLSR protocol. If the MPR received this unfortunate fate, the network will be disconnected given that there is no other cut edge connect the rest of the graph. If one of the edge nodes is gone, the refresh timer for the node will expire and its traces be removed. The rest of the network will continue on by flooding a synchronization topology message throughout the network to notify each node of the changed topology.

There is quite extensive use of threading in our implementation after the boost library timers failed to achieve what we wanted. This opens the possibility for concurrent modification, which we have all had a fair share in dealing. Our prevention technique used mutual exclusion to protect the necessary state data structures required to be accessed and also was used in our producer consumer pattern for receiving messages.

4.4 RFC Compliance

One of our biggest accomplishments in the project has been the degree of compliance to the RFC we've been able to achieve. As mentioned in the result, Snow transmits hello messages, maintains a list of neighbors, transmits transmission control message and computes MPRs in a way very similar to the official specification, differing only by:

1. A full topology routing table specification was skipped over and the functionality was replaced by a simpler map of next hop addresses. We weren't sure at the time of design how OLSR functioned or it's position in routing, we were just three motivated students that took a fancy on Ad-Hoc networking and routing. We didn't notice until later in development that the routing table isn't being used by the actual network's dynamic topology development. It's used by other processes, services, or other entities to find how to get their data to another host. The amount of

forwarding done in our project is to the extent of TC message flooding, none of which require the routing table.

2. In order to manage the complexity of OLSR, Host and Network Association Information base was omitted from our implementation. We felt the scope of our project did not reach the utility function of notifying others about our address with net masks. Another reason why we could not have implemented HNA messages is the fact our implementation runs on hardware address (MAC) routing so masks do not have a part in addressing. Originally for our presentation, we wanted to utilize HNA messages to provide our MANET with a link with the external internet. This could have been done if a node announces itself as the gateway to the internet.

3. Multiple interface address (MID) messages were not useful as we based our project on a single, dongle support interface. These messages were originally created so that one host with multiple wireless interfaces can use one main interface to receive important traffic. They are used to solve the problem where one node will register this host many times since messages can come out of any one of the host's interface. MID unifies these different addresses to associate them with the host's main address. Our project cannot support multiple network interfaces which forces the single dongle to maintain the entire flow of traffic.

4. Addressing in our implementation leveraged MAC hardware addressing rather than the IPv4/6 in the specification. Our understanding of the RFC was that there is an IP address layer above the hardware layer with associations between the MAC and the IP addresses. We left out the IP address layer and stuck with the simple one dimensional address.

In terms of RFC compliance, Snow has achieved far more than what was required by our initial, basic goal. To route messages between three nodes does not require MPR computation at all, let alone the system we have in place. Specifically, for MPR computation, Snow follows the specification exactly by starting with an initial MPR set, computing the degree of each MPR and trimming the MPR set to only nodes which are required. (As specified in section 8.3.1 of RFC 3626)

5. Conclusion

5.1 Summary

In summary, we accomplished what we set out to achieve and then some. Going above and beyond the bare necessity for multi node communication with a fully featured MPR selection and topology forward protocol. Hello and transmission control messages help formulate the backbone of the Snow network by providing dynamic topology construction. Although we did not implement an industry standard implementation, we have all consumed a high degree of ad hoc networking experience and grown comfortable with raw socket messaging. We gathered knowledge on the functioning parts and pieces in network routing and service discovery through implementing our version of OLSR. In hindsight, we wished to have a better understanding and scope of the project at task as many hours were put into making these requirements possible.

5.2 Relevance

The course itself served as the primary inspiration for this project. When trying to figure out something to do we turned to the first unit of the course about MPRs within the OLSR protocol. From there we decided to implement a "essential" version of the OLSR protocol.

Ad hoc networking itself is inherently wireless. With its dynamic network topology and support for mobile agents, ad hoc networking touches on many of the core aspects of the course even aside from the obvious, direct relationship with the ad hoc network routing section of the course.

5.3 Future Work

To build upon this project we plan to continue to follow the OLSR specification outlined in RFC 3626. Currently we have a functioning, though rudimentary, implementation but one that can be expanded upon into a fully compliant OLSR implementation Notably, some of the things we would like to include in future iterations are proper, arbitrary routing between nodes with a complete routing table, MID and HNA messaging to extend the functionality of this network, proper IP address layer in extension to our barebones MAC addressing, and a more robust system that does not crash on rare occasions.

6. Contributions of Team Member

Charlie Li

Handled the hello messages, routing, and MPR computation. Extensive debugging and testing. Helped write technical sections of report.

Kyle Thompson

Wrote much of the report. Handled serializing hello and TC messages. Debugging. Introduced quality of life modern C++ concepts like smart pointers.

Hector Zhang

Handled the tc message, routing computation. Testing and debugging the program. Supporting the detail information and the diagram and help to write the report.

7. References

NS-3 Consortium (2016) NS-3.25 Networking codebase. https://www.nsnam.org/docs/release/3.25/tutorial/ns-3-tutorial.pdf

Andreas (2004) http://www.olsr.org/docs/report_html/node16.html

Andreas (2004) http://www.olsr.org/docs/report_html/node17.html

T. Clausen, P. Jacquet (2003) RFC 3626 (Optimized Link State Routing Protocol)

D. Johnson, Y. Hu, D. Maltz (2007) RFC 4728 (The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks for IPv4)

8. Appendix

8.1 MPR Forwarding (ID: 5d)

Establishing Bi-directionality
Received the message from 1c:bd:b9:7e:b6:6b with originator 1c:bd:b9:7e:b6:6b
Received the message from 1c:bd:b9:88:32:33 with originator 1c:bd:b9:88:32:33
Received the message from 1c:bd:b9:7e:b6:6b with originator 1c:bd:b9:7e:b6:6b
Received the message from 1c:bd:b9:88:32:33 with originator 1c:bd:b9:88:32:33
Forwarding TC Messages
Forwarding the message to 1c:bd:b9:7e:b6:6b with originator 1c:bd:b9:7e:b6:6b
Forwarding the message to 1c:bd:b9:88:32:33 with originator 1c:bd:b9:7e:b6:6b
Forwarding the message to 1c:bd:b9:7e:b6:6b with originator 1c:bd:b9:7e:b6:6b
Forwarding the message to 1c:bd:b9:88:32:33 with originator 1c:bd:b9:88:32:33
Forwarding the message to 1c:bd:b9:7e:b6:6b with originator 1c:bd:b9:88:32:33

8.2 Node Forwarding (ID: 6b)

Establishing Bi-directionality
Received the message from 1c:bd:b9:7e:b6:5d with originator 1c:bd:b9:7e:b6:5d
Received the message from 1c:bd:b9:7e:b6:5d with originator 1c:bd:b9:7e:b6:5d
No forwarding messages

8.3 Node Forwarding (ID: 33)

Establishing Bi-directionality
Received the message from 1c:bd:b9:7e:b6:5d with originator 1c:bd:b9:7e:b6:5d
Received the message from 1c:bd:b9:7e:b6:5d with originator 1c:bd:b9:7e:b6:5d
No forwarding messages

8.4 MPR Verbose Log (ID: 5d)

Sent a TC message
Sent a hello message

Received the message from 1c:bd:b9:7e:b6:6b with originator 1c:bd:b9:7e:b6:6b
Process hello message and update state
Found a new neighbor from hello msg
Determined the state is asymteric
Update the expire time of the link as soon as if bi directional time is greater
Schedule a 7 seconds to do expire this link
Number of neighbors detected on this message from 1c:bd:b9:7e:b6:6b is 0
Compute the MPR base on the neighbor and 2-hop neighbors and MPR set

Number of two hop neighbors is 0

Received the message from 1c:bd:b9:88:32:33 with originator 1c:bd:b9:88:32:33
Process hello message and update state
Found a new neighbor from hello msg
Determined the state is asymteric
Update the expire time of the link as soon as if bi directional time is greater
Schedule a 7 seconds to do expire this link
Number of neighbors detected on this message from 1c:bd:b9:88:32:33 is 0
Compute the MPR base on the neighbor and 2-hop neighbors and MPR set
Number of two hop neighbors is 0

Sent a hello message

Received the message from 1c:bd:b9:7e:b6:6b with originator 1c:bd:b9:7e:b6:6b
Process hello message and update state
We have gotten a hello message form this neighbor before, updating expire time
Determined the state is bidirectional
Update the expire time of the link as soon as if bi directional time is greater
Process the advertised neighbors
The address 1c:bd:b9:7e:b6:6b has chosen me as an MPR
Compute the MPR base on the neighbor and 2-hop neighbors and MPR set
Determined 1c:bd:b9:7e:b6:6b is a two hop neighbor from 1c:bd:b9:88:32:33
Number of two hop neighbors is 0

Received the message from 1c:bd:b9:88:32:33 with originator 1c:bd:b9:88:32:33
Process hello message and update state
We have gotten a hello message form this neighbor before, updating expire time
Determined the state is bidirectional
Update the expire time of the link as soon as if bi directional time is greater
Process the advertised neighbors
The address 1c:bd:b9:88:32:33 has chosen me as an MPR
Compute the MPR base on the neighbor and 2-hop neighbors and MPR set
Determined 1c:bd:b9:88:32:33 is a two hop neighbor from 1c:bd:b9:7e:b6:6b
Number of two hop neighbors is 0

Creating hello message and adding link 1c:bd:b9:7e:b6:6b who choose me as a MPR candidate
Creating hello message and adding link 1c:bd:b9:88:32:33 who choose me as a MPR candidate

Received the message from 1c:bd:b9:88:32:33 with originator 1c:bd:b9:88:32:33
Received the message from 1c:bd:b9:7e:b6:6b with originator 1c:bd:b9:7e:b6:6b

Process tc message and update state
Check the neighbor of this message if it is not originator else discard
Check the tc message and discard if message received out of order
Clean all older topology tuple set

Process the advertised neighbors
Forwarding TC packet: True
TC message forwarding was successfully sent

8.5 Node Verbose Log (ID: 6b)

Sent a TC message
Sent a hello message

Received the message from 1c:bd:b9:7e:b6:5d with originator 1c:bd:b9:7e:b6:5d
Process hello message and update state
Found a new neighbor from hello msg
Determined the state is asymteric
Update the expire time of the link as soon as if bi directional time is greater
Schedule a 7 seconds to do expire this link
Number of neighbors detected on this message from 1c:bd:b9:7e:b6:5d is 0
Compute the MPR base on the neighbor and 2-hop neighbors and MPR set
Number of two hop neighbors is 0

Received the message from 1c:bd:b9:7e:b6:5d with originator 1c:bd:b9:7e:b6:5d
Process hello message and update state
We have gotten a hello message form this neighbor before, updating expire time
Determined the state is bidirectional
Update the expire time of the link as soon as if bi directional time is greater
Process the advertised neighbors
Compute the MPR base on the neighbor and 2-hop neighbors and MPR set
Determined 1c:bd:b9:88:32:33 is a two hop neighbor from 1c:bd:b9:7e:b6:5d
Number of two hop neighbors is 1
Check all the two hop neighbors that have been covered by an MPR with address 1c:bd:b9:88:32:33
Adding to MPR selector 1c:bd:b9:7e:b6:5d

Creating hello msg and setting 1c:bd:b9:7e:b6:5d who choose me as a MPR candidate

Received the message from 1c:bd:b9:7e:b6:5d with originator 1c:bd:b9:88:32:33
Process tc message and update state
Check the neighbor of this message if it is not originator else discard
Check the tc message and discard if message received out of order
Clean all older topology tuple set
Process the advertised neighbors
Forwarding TC packet: False

Create tc message
Setting message header
Adding 1c:bd:b9:7e:b6:5d to TC mpr selector
Sent a TC message

8.6 Node Verbose Log (ID: 33)

Sent a TC message
Sent a hello message

Received the message from 1c:bd:b9:7e:b6:5d with originator 1c:bd:b9:7e:b6:5d
Process hello message and update state
Found a new neighbor from hello msg
Determined the state is asymteric
Update the expire time of the link as soon as if bi directional time is greater
Schedule a 7 seconds to do expire this link
Number of neighbors detected on this message from 1c:bd:b9:7e:b6:5d is 0
Compute the MPR base on the neighbor and 2-hop neighbors and MPR set
Number of two hop neighbors is 0

Received the message from 1c:bd:b9:7e:b6:5d with originator 1c:bd:b9:7e:b6:5d
Process hello message and update state
We have gotten a hello message form this neighbor before, updating expire time
Determined the state is bidirectional
Update the expire time of the link as soon as if bi directional time is greater
Process the advertised neighbors
Compute the MPR base on the neighbor and 2-hop neighbors and MPR set
Determined 1c:bd:b9:7e:b6:6b is a two hop neighbor from 1c:bd:b9:7e:b6:5d
Number of two hop neighbors is 1
Check all the two hop neighbors that have been covered by an MPR with address 1c:bd:b9:7e:b6:6b
Adding to MPR selector 1c:bd:b9:7e:b6:5d

Creating hello msg and setting 1c:bd:b9:7e:b6:5d who choose me as a MPR candidate

Received the message from 1c:bd:b9:7e:b6:5d with originator 1c:bd:b9:7e:b6:6b
Process tc message and update state
Check the neighbor of this message if it is not originator else discard
Check the tc message and discard if message received out of order
Clean all older topology tuple set
Process the advertised neighbors
Forwarding TC packet: False

Create tc message
Setting message header
Adding 1c:bd:b9:7e:b6:5d to TC mpr selector
Sent a TC message