Jake Luoma
Network Systems Spring 2020 Final Project Report

# Introduction

Search and Rescue (SAR) operations are frequently conducted in wilderness locations with little to no cell tower coverage, limiting the team's access to the internet. Further, rugged geography and poor line of sight causes assets in the field to frequently go out of contact, hampering communication and exchange of information. Under such conditions, data transmitted in a traditional network is unlikely to propagate successfully to the recipient. Additionally, when controlling assets like drones, line of sight is generally required, limiting their utility. Limited ability to communicate is one of the biggest challenges for SAR operations today. If this problem were to be solved, coordination of teams in the field, propagation of new evidence, and ultimately time to find the subject could all be improved.

To solve these challenges, a solution must allow for nodes to engage in opportunistic and cooperative networking, delivering data to recipients when a path is available and storing the data for future transmission when one is not. Additionally, the network should be highly robust in the face of missing nodes and broken links. One reasonable solution is to implement a mesh network and outfit the team with compatible radios. In fact, there are hardware and software platforms which implement this exact functionality (for example GoTenna in concert with ATAK). However, GoTenna is a very expensive hardware solution (far beyond the budget of most SAR teams) and ATAK does not have SAR-specific software. The field could benefit from a software-defined networking/radio solution which could be run on a consumer-grade cell phone or radio.

This project does not tackle the hardware challenges associated with that, but instead implements a mesh network using OpenFlow. It is intended less as a research project and more as a chance for me to learn about software-defined networking in the context of a real problem.

This work shows that a mesh network can be implemented with software-defined networking. I have produced a controller which has functionality including node discovery, route selection, intra-network transmissions, and inter-network transmissions. Additionally, I have implemented node discovery and route selection in a unique single step, which should enable network resilience and responsiveness in the face of broken links.

# Related Work

The metric used for path prioritization and selection is very important. In a wireless network, some packets will not make it to the recipient due to various effects like interference and decreasing transmission power with distance. One of the earliest metrics to support path selection is the *expected transmission count* (ETX)[1]. In the original version, the network undergoes a characterization phase in which the ETX is characterized along any given path. This ETX calculation is bi-directional, as it includes the number of transmissions for both sending a packet and receiving a response. In this project, I use a

[1] De Couto, D., Aguayo, D., Bicket, J., Morris, R. A High-Throughput Path Metric for Multi-Hop Wireless Routing. MobiCom. September 14-19, 2003.

uni-directional ETX value (sending the packet only) for path selection. The most important distinction is my solution constantly updates ETX values as the network operates as a side-effect of sharing routing tables. This makes my implementation more responsive to changes in the network topology.

Unicast transfers in a multi-hop network can be further improved through a technique like ExOR[2], which takes advantage of transmissions which fortuitously and probabilistically travel to more distant nodes in the network. The system I've implemented also uses unicast transfers but does not take advantage of transmissions which travel further than expected. In fact, the simulation, as implemented, assumes that such transmissions are not possible.

An alternative approach is to use a routing technique like Batman[3], in which no single node has knowledge about the optimal path to the other nodes in the network. This obviates the need to propagate network changes to each node, further increasing efficiency. At a high level, this is done by making each node aware of the direction from which it received a packet (or multiple copies of the same packet) and it makes routing decisions using that information. This approach also obviates the need to characterize a metric like ETX, further increasing efficiency. My solution does not take this approach, as it is more complicated to implement.

# System Design

The system has several components. First is the simulation tool, which impacts the other components due to assumptions made to accommodate the simulation software. Second is the neighbor discovery and path selection protocol. Third is the protocol for intra-network communication. And finally, is the protocol for inter-network communication.

## Simulation

Mininet was selected to drive the simulation with POX as the OpenFlow controller. There are several advantages and disadvantages to using Mininet.

The advantages are that a network topology can easily be defined, it can be modified during the simulation, hosts are shell processes which makes running custom scripts easy, and switches can be programmed with OpenFlow. Therefore, much of the functionality to accurately simulate a mesh network is present.

The primary disadvantage is that Mininet can only simulate ethernet links, not wireless broadcasts.

As an alternative, I evaluated Mininet-Wifi, a fork of Mininet which adds wireless links and access points to the base model. This would be more appropriate for simulating a mesh network. However, communication would have to be mediated between access points, which requires hardware support that my laptop doesn't have.

---

[2] Biswas, S., Morris, R. ExOR: Opportunistic Multi-Hop Routing for Wireless Networks. SIGCOMM. August 21-26, 2005

[3] Batman website: https://www.open-mesh.org/projects/batman-adv/wiki/BATMAN_V

Therefore, I used the base Mininet and ethernet links with variable packet drop rates to simulate wireless communication. I also assumed that individual nodes are capable of unidirectional transmission and omni-directional receive.

The network is represented as a set of hosts, each connected to their own switch with lossless ethernet links. The switches are then connected to each other with lossy ethernet links. The switches all share a single controller. This is shown in Figure 1. Each host-switch pair is logically treated as a single unit – meaning that the controller treats each switch as if it is on a particular host when generating routing rules. The network is represented in this fashion to allow for customization of network behavior by programming the switches with OpenFlow.

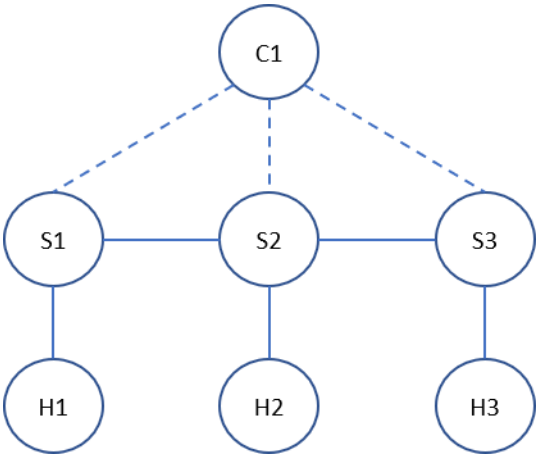


*Figure 1: A sample topology in Mininet*

To enable the creation of host-switch pairs, special behavior is required from both the hosts and the controller when the simulation is started. The hosts each have an entry added to their ARP tables which contains a reserved IP-mac address pair. This is done to avoid ARP requests by the host when sending a packet to that address pair. Then, a custom server program is started on each host which sends a packet to the reserved IP address. The host's associated switch receives the packet and forwards it to the controller. The controller recognizes that the packet was sent to the reserved IP address and therefore associates the host's IP/mac address with the switch that received the message, thus forming the host-switch pair. The host is added to each switch's routing table as a special entry indicating that it is a connected host.

## Neighbor Discovery and Path Selection

At the start of the simulation, hosts are not aware of each other and ARP functionality is not provided. Therefore, the hosts must become aware of each other through other means. Additionally, paths from host to host must be identified and flow rules must be pushed to the switches.

Neighbor discovery and path selection are implemented in a single step. The controller has two routing tables for each switch. A primary routing table contains entries which associate IP address, mac address, ETX, port, and whether the host has internet access. There is one entry per IP address. Additionally, the controller has a secondary routing table for each switch. This secondary routing table is the same as the

primary routing table but can have multiple entries per IP address. The secondary tables always contain an entry for the relevant connected host. They are used to gather information about reachable hosts and then update the primary routing tables with only the best paths.

A timer is set to occasionally go off on the controller, prompting it to, for each switch, calculate ETX values for each entry in the secondary routing table, cull them to leave only the best entry for each IP address, and then update the primary routing table. The secondary routing table is then cleared. The controller prompts each switch to send its primary routing table to its connected host so the host can add newly discovered IP/mac addresses to its ARP table. This enables hosts to send packets directly to other known hosts. Additionally, updated flow rules are pushed to each switch based on the new primary routing tables.

Then, the controller tells each switch to flood its primary routing table out all ports. This simulates a directional transmission that sweeps a sphere around the switch. In the current implementation, the routing table's size is limited to the capacity of a single packet. The packet is replicated 5 times to support the generation of ETX data.

A receiving switch forwards each packet to the controller. The controller updates the secondary routing table, either adding a new entry if a routing table from that port has not been received yet or updating an existing entry by incrementing its received count. This information is collected until the next time the timer goes off. The process then repeats.

## Intra-Network Communication

Intra-network communication is performed from host to host using TCP. Each host tracks the IP addresses and mac addresses of hosts it has become aware of in both its ARP table as well as in the server program it is running. This enables a host to send data directly to any of these hosts. If there is a path to the destination, the network will forward the packet via the best available route thanks to the process outlined in the Neighbor Discovery and Path Selection section. If no path is available, the packet will be sent to the controller which will store the packet for a short time in the hope that a path becomes available. The first time a packet is seen by a switch, it is sent to the controller, which then pushes a flow rule to the switch and forwards the packet.

## Inter-Network Communication

The IP addresses 192.168.100.xxx are reserved for hosts within the network. A packet sent to an IP address outside of that range is assumed to be destined for the internet and is routed to the nearest host/switch pair that has internet access. On simulation start, at least one of the hosts is manually defined as having an internet connection. At the controller level, a list of host/switch pairs with internet connections is maintained. When a packet destined for the internet is sent to the controller, this list is consulted, and, for each entry in the list, the switch's primary routing table is consulted to identify the closest one. The packet is then forwarded to that one. On receipt of the packet by the switch with internet access, a message is written to the log that the packet was received. The controller is therefore involved for every hop to the switch with an internet connection. This could be made more efficient by pushing flow rules to the switch.

## Evaluation/Findings

There are several important tests for evaluating a system like this.

1. Time for optimal routes to be identified (settling time) at the start of simulation
2. Settling time after a topology change
3. Selection of the correct routes when links drop a certain fraction of packets
4. Functionality of intra-network communication – can hosts ping each other? Do packets get lost after a topology change?
5. Functionality of inter-network communication – can a packet make it to the internet?

Since path selection is performed as a batch process, the settling time metrics are evaluated in terms of the number of refreshes (number of times the controller timer goes off). As currently configured, a refresh occurs every 3 seconds. These tests evaluate the performance of each major component as detailed in the System Design section.

### High Level Methodology

As discussed in the System Design section, simulations are performed using Mininet. Two topologies are used to conduct the tests: a chain topology and a pentagon topology. The chain topology is of length n, containing n host/switch pairs, as shown in Figure 2. The pentagon topology contains 5 host/switch pairs as shown in Figure 3. Unless otherwise stated, all links deliver all packets. When verifying settling time, I wrote the routing tables to logs at each refresh and manually verified when the tables had settled. In the other experiments, the commands used to run the experiment are present in the screenshots.

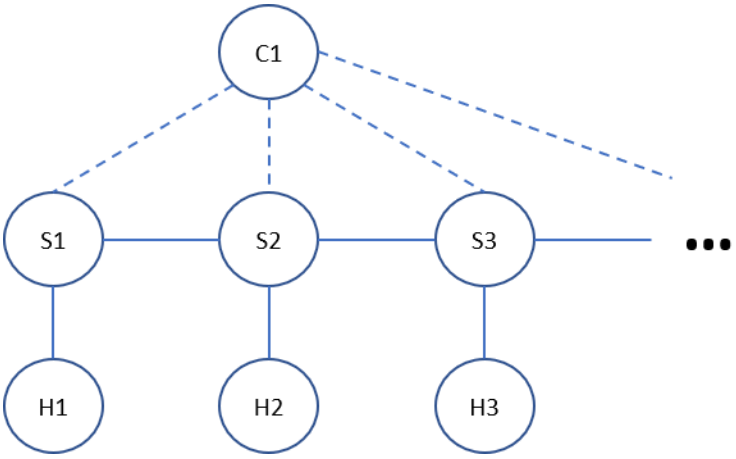


*Figure 2: An example of the linear topology*

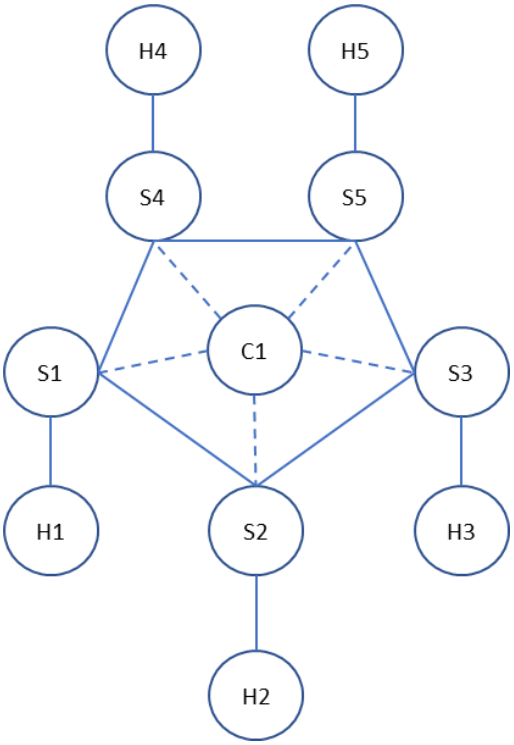


*Figure 3: An example of the diamond topology*

## Settling Time at Simulation Start

The chain topology is evaluated at different lengths to identify the settling time.

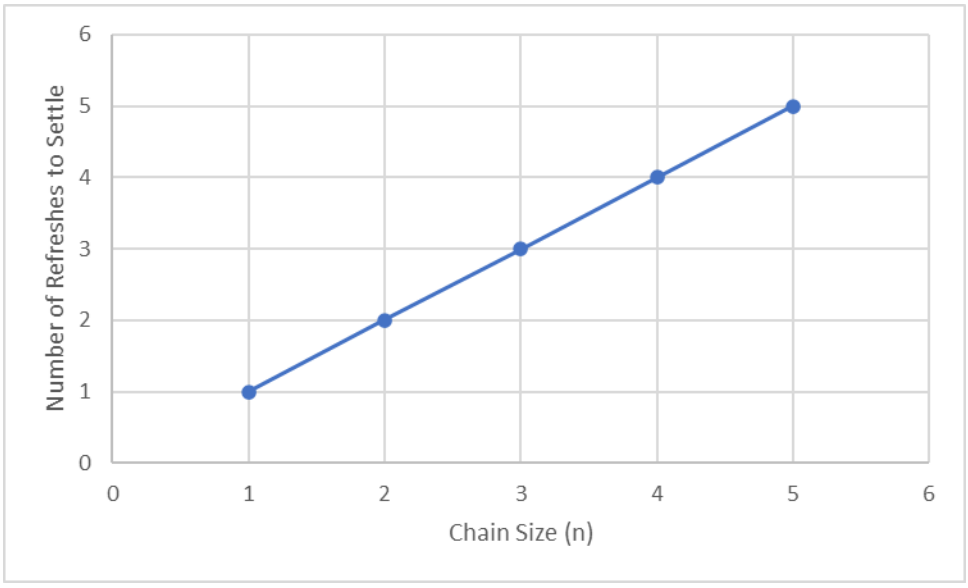


*Figure 4: The number of refreshes required for a chain network of various sizes to settle*

As can be seen in Figure 4, the settling time depends linearly on the length of the chain, requiring a number of refreshes equal to the chain length to settle. This is expected, as information about a particular node can propagate one hop per refresh.

## Settling Time after a Topology Change

The chain topology is evaluated at different lengths. It is allowed to settle and then the last link is broken.

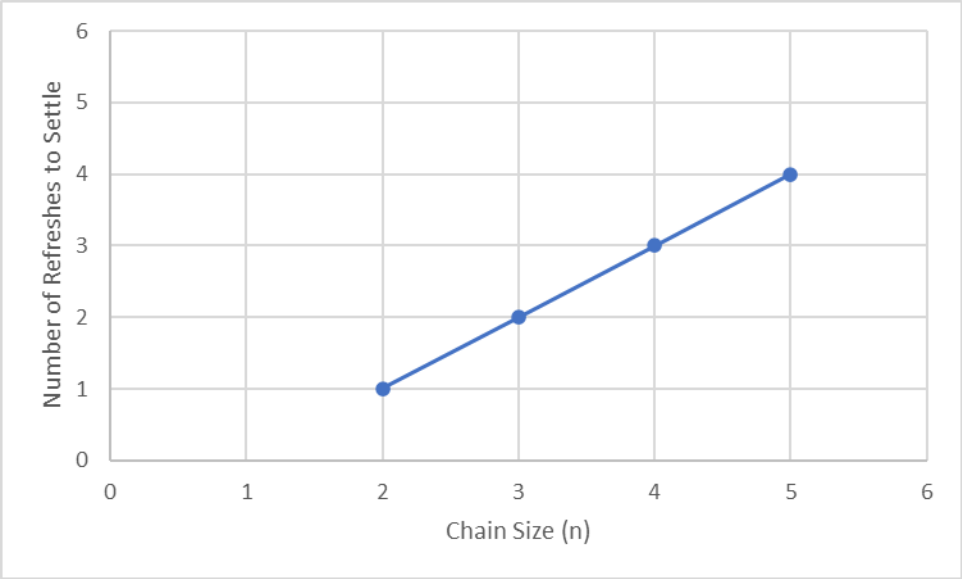


*Figure 5: The number of refreshes required for a chain topology to settle after the last link is broken*

The length of the remaining longest chain is equal to n-1, suggesting that the settling time will be n-1 refreshes. This is what was observed.

The situation is a bit different with the pentagon topology. Imagine that the system has settled and then the link between S2 and S3 is broken. It only takes 2 refreshes to settle onto the optimal path of S2 -> S1 -> S4 -> S5 -> S3. The reason is that, right before breaking, the optimal path for S4 is S4 -> S5 -> S3. It therefore already knows part of the optimal path. The distance from S4 to S2 is only two hops, so it will only take 2 refreshes for this information to reach S2.

## Selection of the Correct Route when Links Drop Packets

Effectively, this is a test of whether the ETX calculation is correct. In the pentagon topology, with two paths from S1 to S3, the best path depends on the ETX for each path. If no links except for S2 -> S3 drop packets and that link drops 80% of packets, the ETX from S1 to S3 is 3 along the top path and 6 along the bottom path. Therefore, the top path should be selected. I don't have charts to illustrate this, but it is indeed what is observed.

## Functionality of Intra-Network Communication

In this experiment I used the pentagon topology. The network was allowed to settle and then the link between S2 and S3 was broken. If I waited for two refreshes to occur and then had H2 ping H3, all pings were successful. However, if I downed the link and immediately had H2 ping H3, the pings were lost. This occurs because it takes time for S2 to recognize that S3 has become unreachable (as implemented it takes 1 refresh). During that time, S2's routing table still indicates that S3 is reachable. If a packet arrives during that time, it is forwarded according to the routing table onto the downed link and is lost. It is therefore important that packets be sent with TCP when using the controller as implemented in order to handle link loss.

This behavior is demonstrated in Figure 6 below: the system is allowed to settle, the link between S2 and S3 is broken, H2 immediately attempts to ping H3 (and fails), and then H2 waits for two refreshes before trying to ping H3 again (and succeeds).

```
mininet@mininet-vm:~/mininet/mininet$ sudo /usr/bin/python2.7 /home/mininet/mininet/mininet/mininet-mesh-topology.py
Unable to contact the remote controller at 127.0.0.1:6653
mininet> link s1 s3 down
src and dst not connected: s1 s3
mininet> link s2 s3 down
mininet> h2 ping -c1 h3
PING 192.168.100.3 (192.168.100.3) 56(84) bytes of data.

--- 192.168.100.3 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet> h2 ping -c1 h3
PING 192.168.100.3 (192.168.100.3) 56(84) bytes of data.
64 bytes from 192.168.100.3: icmp_seq=1 ttl=64 time=430 ms

--- 192.168.100.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 430.211/430.211/430.211/0.000 ms
mininet>
```

*Figure 6: Demonstration of packet loss after a link is downed and before flow rules on the switches expire*

## Functionality of Inter-Network Communication

It should be possible to send a packet outside of the IP address space reserved for members of the network. This functionality is operational, as demonstrated in Figure 7 below:

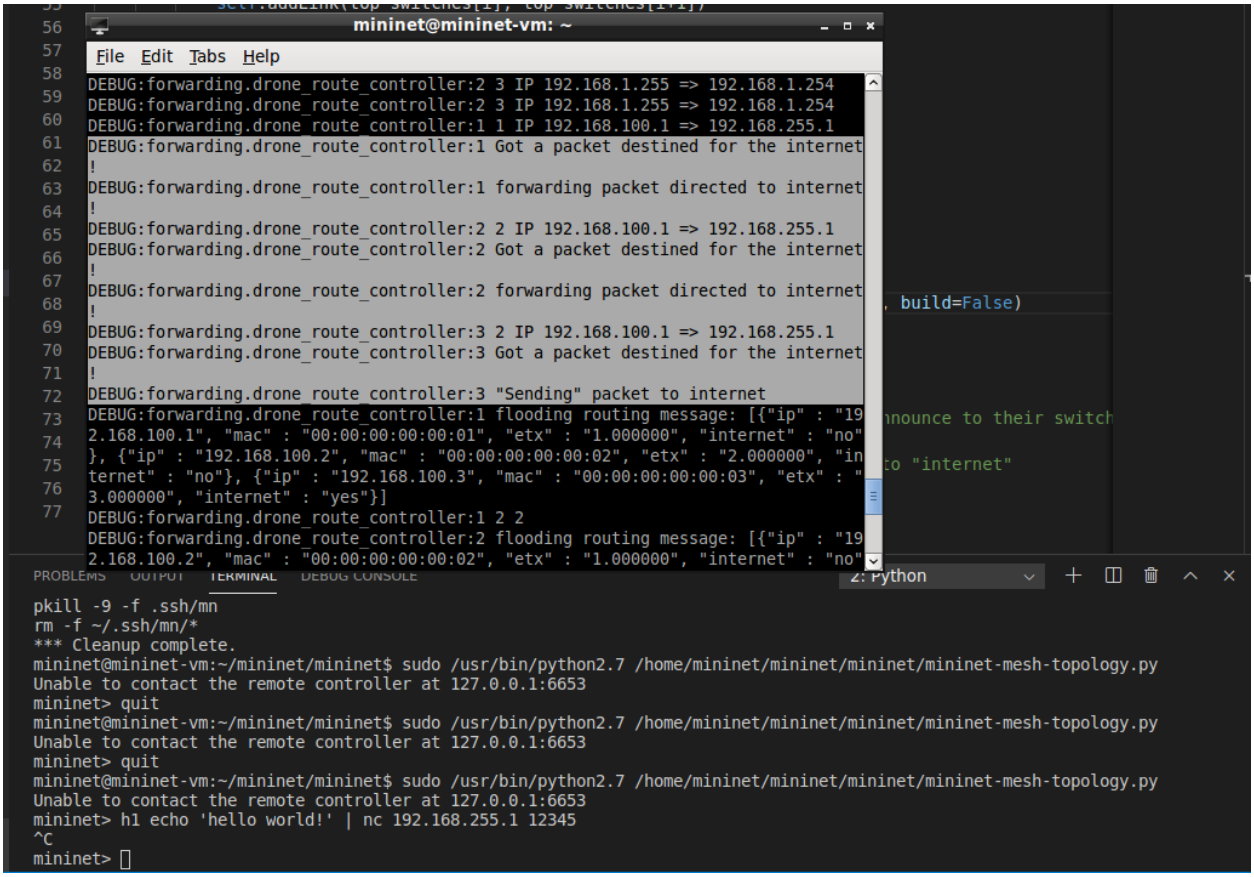


*Figure 7: Packets can be sent and automatically routed to the internet*

## Review of Team Member Work

Since this was a solo project, I did all the work. By the first checkpoint, I had tried several simulation tools and decided on Mininet. By the second checkpoint, I had completed implementation of most simulation functionality, other than inter-network communication. By the final project due date, I completed system evaluation, the presentation, and this report.

## Conclusions and Improvements

On starting up a network with a chain topology, the system takes a number of refreshes equal to the chain length before it settles on the optimal paths. This raises an interesting question: if there are multiple routes through a topology, how long will it take to settle? Take the pentagon topology; assume the link quality along the bottom path is extremely poor, so the optimal route from S1 to S3 is along the top path, with a chain length of 4 nodes. This implies that the settling time will be 4 refreshes, which is exactly what I observed when testing. This therefore suggests that the number of refreshes for a topology with multiple paths and packet loss to settle is equal to the chain length of the longest, best path.

The situation is better once the optimal paths have been identified and then the network changes (say, a link goes down). Since some switches are along the new optimal path and already have information about the optimal path, the settling time is reduced. The amount by which it is reduced depends on the length of the chain through which this information must propagate.

The system as designed performs the sharing of routing tables, generation of ETX data, and path selection all in a single step. This process has the advantage of making for a network which responds faster to changes in link quality than the original ETX paper. That said, the current process is limited to small routing tables (<100 entries since the routing table must fit in a single packet) and is performed as a batch process. The network would be far more responsive if this were run as a continuous process where routing tables are shared every 0.1 seconds, ETX is tracked as a moving average over the past second, and addition or removal of entries from the primary routing table are propagated immediately. The settling time would still be limited by the number of hops the information must travel, but these changes would ensure the network responds rapidly to routing updates.

One limitation that would remain is that it would take time for the network to recognize when a link goes down (in the new proposed implementation it would take 1 second). Therefore, as observed in the experiments above, packets could still get lost after a link failure. As a result, host applications would still need to implement some logic to handle lost packets.

Another limitation that would remain is that the routing table would need to fit in a single packet. To increase the supportable size of the network, the routing table could be split into multiple numbered packets. Only parts of the routing table would be sent per time period. This would somewhat slow down routing table propagation but would enable larger tables. The network would still likely be functionally limited to fewer than 1000 nodes, however. An alternative is that, at each refresh, only diffs of the routing table are sent (added/removed entries, changes in ETX). If a new node comes into contact it could request the full table. To avoid nodes sending the entire table every time, entries with updated ETX values could sent on a less frequent interval. This would greatly increase the supportable size of the network.

Finally, packets could incorporate some kind of versioning to improve caching behavior. If a cached packet contains data which is superseded by data in a more recent packet, versioning would ensure that only the more recent packet is kept and forwarded, simplifying application logic.