The burden and information gain of path-based topology discovery

Remi Hendriks
University of Twente
PO Box 217, 7500 AE Enschede
the Netherlands
r.hendriks@student.utwente.nl

ABSTRACT
There exist many routing algorithms for networks that allow nodes to efficiently send messages through a network, these algorithms require a map of the network. One method for gaining knowledge about the topology of a network is to store the path a packet has taken and include it in the packet header. This will burden the network with the extra required overhead to share the paths of packets, however a lot of information can be gained from analyzing this data. This paper will analyze the information gain and burden of this method. To accomplish this, we will write software that examines received packets from a simulated network and updates routing tables based on the paths these packets took. Statistics about information gain will be examined and presented.

Keywords
Path-based discovery; information gain; metadata burden on network; network topology; topology discovery.

1. INTRODUCTION
Networks are continuously growing in size as the number of devices in our society keeps increasing, as well as the size of incoming and outgoing packets for these devices. To ensure that the networks can scale with the increase in size and traffic, routing algorithms must be as efficient as possible. The goal of these algorithms is to send data from sender to receiver, to accomplish this they must add data to the packets that is not required by the receiver but required by the network to operate. This is called metadata, metadata contains information about destination, source, packet size, etc. depending on the network design.

This metadata is very valuable for routing algorithms, as they can use it to build a map of the network and use it to decide where to send packets to. The path a packet took through the network is extremely useful information that can be added to the metadata of the packet structure. This information tells routing algorithms a lot about what the network looks like. However, they require a lot of metadata to be send through the network. This added overhead is undesired.

This paper aims to; discuss the up- and downside of adding paths to the packet structure, create software that realizes routing maps being built based on packet paths, and analyze the effects this has on a network.

28th Twente Student Conference on IT, Jan. 29th, 2021, Enschede, The Netherlands.

2. BACKGROUND
To understand the aims of this paper, we must first give some background on routing, and provide information about the simulator that will be used.

2.1 Routing algorithms
To ensure that packets get delivered to their destination routing algorithms are used, this can be new packets or packets that are being forwarded. Ideally the routing algorithm can find the best path for all packets being sent through the network. To realize this, however, the routing algorithm would need a good map of the network. This map of the network is especially difficult when the network is constantly changing, which is often the case. Existing connections between nodes can drop, new connections can be found, or perhaps the nodes in the network change locations.

If every node in the network would flood the network with their own neighbors, each node would be able to build a complete map for all connected nodes. However, this would require a lot of information to flood the network if the network is non-static then this information would have to be flooded many times. This is undesired as it would make the network less efficient. Packet loss will be more likely, packets will take longer to process, computational requirements will increase. Therefore, it is important that all routing algorithms have an efficient method of mapping the network.

2.2 The simulator
Before we can write a program for our purposes, we must first simulate a network. To realize this, we will use 'The Opportunistic Network Environment Simulator' ('The ONE'). This is a simulation environment that we will use to simulate a network which we can later use to test our software on and analyze. The network I will be simulating can be seen below (figure 1).

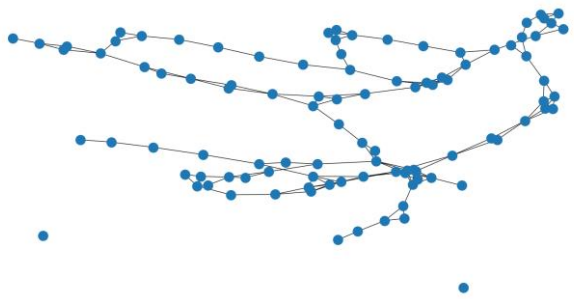

Figure 1. A map of the network

The network is 100 nodes in size and on average each node has 3.14 edges. The simulation uses epidemic routing to flood this network with packets, where packets will be dropped if they have been received previously. The epidemic routing algorithm implemented in the simulator replicates messages and sends

these copies to all neighbors, except the one he received the original from. This ensures that any node with a connection to a node, that has created a new packet, will receive that packet and will only receive it once. The simulation outputs information to a log where we can find the creation of the edges at initialization, and we can see every time a message got sent to a node.

Whilst the network consists of 100 nodes, 2 of these nodes do not have any neighbors and thus are not connected to the network. Therefore, these two nodes will not be included in any results and the analysis thereof. The network will be seen as having 98 nodes total.

2.3 Python libraries

For this project we use the following Python libraries: Matplotlib, and Networkx.

Matplotlib allows us to import many mathematical functions that we use to parse through the data and analyze the results, it is also used to generate and show graphs. Networkx is a library that is used to represent networks in Python, in this case the network from the simulator.

3. RELATED WORK

Breitbart et al. [1] discussed the importance of topology information in networks and developed novel algorithms for discovering topology in heterogeneous IP networks.

Donnet et al. [2] did a survey on measurements of the network topology. Diaz et al. [3] discuss the overhead of a topology discovery algorithm and talk about the related trade-offs.

4. RESEARCH GOAL

The goal of this research is to find out whether keeping track of the path packets took to reach its destination can be used as a networking mapping approach. More specifically, does the burden of having to increase packet size for this information justify the effectiveness this has on mapping the network. To find these answers we must create a program that can map the network through analyzing paths and examine the result to determine the validity of this solution.

4.1 Research question

Can a network be mapped by keeping track of paths that packets take in a network, and is this a viable solution?

5. METHODOLOGY

To approach the research question, a Python program was created. In this section we will discuss the development of this program.

5.1 Translating the simulation

Firstly, we translate the graph into our Python program. For this we create the nodes and get their neighbors from the log. Once all this information is parsed, each node will have a tiny map of the network that consists of their direct neighbors.

Next, we go through the logs and look at all the successfully received messages, filtering out received messages that had already been received prior and thus were dropped. For each received message, all that is captured in the log is: the timestamp, the message ID, the sender, the receiver, and whether it was a successfully received message.

If a message is a newly created message, we give it a path that contains the origin and destination of this message. If we find a message that the origin forwarded, we look through the events from the origin node to find the event where the origin received this packet. We then copy the path from there and append it with the current destination. This ensures that for each message received event we will have the path this message took so far.

5.2 Mapping the network

Now that we have the path for each received message event, we can create for each node a map of the network, based on the paths of the messages the node received. The network map for each node has the following structure:

[(Node_ID, [Neighbor_IDs..])]

This is a list of tuples, where each tuple has a node coupled to a list of that node's neighbors.

To update our network map based on a path, we loop over the path starting with the first node. We check whether we have this node in our network map. If we do not, we add it to our map and, if they exist, we add the previous and next node in the path to this new node's neighbor list. If we do have this node in our network map, we check whether we have the previous and next node in the path, in the current node's neighbor list and add them if they are not yet in that list.

5.3 Getting useful data

For us to examine the validity of this network mapping method, we must keep track of data that show the efficiency. For these reasons we will save for each node: the total amount of received messages, the message IDs that updated the routing table together with what event number this was.

Finally, when all events have been parsed, we can examine for each node how accurate their map is. We do this by comparing what a complete network map looks like and what the network map of an individual node looks like.

5.4 Reducing metadata

Mapping the network comes at a cost; packets in the network are carrying more metadata to store the path they have taken. This metadata is undesired and should be as little as possible. Therefore, we will quantify the overhead in our program to determine the total shared metadata. Finally, we will change the number of hops stored in our paths. We will either store all hops, or the hops up to n. For these different configurations we will determine the efficiency, by comparing the overhead cost to the completeness of the network maps.

We could also look at storing the last n hops, however this will only limit the vision of the nodes. This is because a node will then only be able to see n hops far, as any node beyond that hop count will not be stored in the path.

6. ANALYSIS

With the data from the simulation being parsed through the Python program, we have gained information on path-based network mapping. We can use this data to analyze the information gain and burden of this approach.

6.1 The completeness of the network maps

The size of all neighbor lists combined is 314, for every edge in the network there are two entries in the combined neighbor list (one for each node). This means that the network has 157 edges, and 100 nodes. Due to the randomness of the network, two nodes are not connected. This means that the connected part of the network consists of 98 nodes.

Each node in the entry ended up with having an entry in their map for every connected node in the network. However, for each node to have a full map of the network, they would need to have neighbor lists that collectively equal 314 in size. However, the average neighbor list size is 305.95. This means that each node is unaware of 4.03 edges out of 157 on average.

On average each connected node was aware of 97.44% of all the edges in the network. The 2.56% that a node is unaware of are most likely edges that have a high travel time or are outperformed severely by other edges. Thus, these edges have little value, or even no value at all, to this node.

Since the number of packets flooded into the network is finite, each node received 3,637 messages on average, the network maps of nodes are incomplete. If there were infinite number of packets being flooded, then eventually all nodes will have a full map of the network.

6.2 Differences in learning between nodes

Some nodes learn the layout of the network faster and with fewer messages than others. The nodes in the center of the network have a lower average distance between nodes than the nodes in the edges of the network. It is expected that those nodes in the center will learn about the network faster than those at the edges. There may also be difference in learning when comparing nodes with few neighbors and nodes with many neighbors.

However, when comparing the average time needed for nodes to learn 25, 50, and 75% off the network map based on the number of neighbors. We find that there is no statistically significant difference in learning time when looking at the neighbor count.

Table 1. Learning time (in seconds) for nodes with 2, 3, 4, or 5 neighbors to learn 25, 50, and 75% of the network.

	25%	**50%**	**75%**
2	119	463	1995
3	115	452	1973
4	119	433	1994
5	132	454	1963

Above you can see a table showing the timestamp in the simulation where the node has received enough data to have learned 25, 50, or 75% of the network. The average of nodes is taken based on their neighbor count; 2, 3, 4, or 5. This data disproves that nodes with higher number of neighbors learn faster compared to those with lower number of neighbors when it comes to the learning algorithm discussed in this paper.

6.3 The information gain

On average each node received 3,637 messages, of which 76.03 gave new information. This shows that on average 3,560.97 messages (98% of all messages) did not give new information about the network. This is a significant amount, and the information gain is quite low.

From observing the message IDs that gave new information you can see that IDs with small values are very likely to give new information, whereas IDs with high values are very unlikely to give new information. This is because as the IDs increase the nodes have received increasingly more messages. When a node has already received many messages, that node will have a good map of the network and will be unlikely to gain new information from future messages.

This can also be seen when looking at the timestamp when messages are received, at low timestamps more packets containing new information get received than at higher timestamps. In the bar graph (figure 2) you can see the number of packets, with new knowledge about the network, that were received by nodes within certain intervals.

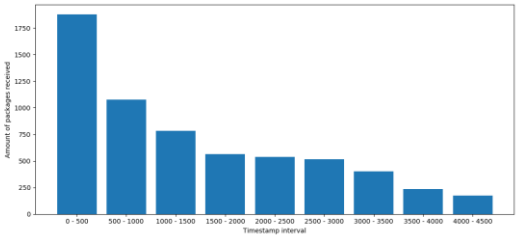

Figure 2. The number of packages that gave new information for various time intervals (in seconds).

6.4 The burden

As the hop count of a packet increases, the number of nodes in its path increase too. This means that the metadata required can become very large if the packet had to move through many nodes in the network. In the worst case the path size would be equal to the total amount of connected nodes, however it is very unlikely that a packet would visit every node in the network.

When assuming that each address in the path takes up a single byte, then we find that the combined metadata for every successfully received message is equal to 4,418,194 bytes (4.4GB). The combined metadata of all received messages that gave new information to nodes is 104,863 (0.1GB), which is 2.4% of the total combined metadata.

6.5 Limiting the path size

Table 2: Data for path sizes 5, 10, 15, 20, and 25.

	5	**10**	**15**	**20**	**25**
Overhead	1.715	3.032	3.853	4.239	4.377
Completeness	0.973	0.974	0.974	0.974	0.974
75% time	2528	2153	2041	1984	1981

The results for limiting the path size can be seen in table 2. In this table ‘Overhead’ is the combined metadata of all packets successfully received (in GB), ‘Completeness’ is the average completeness of the network map, ‘75% time’ is the average amount of time needed for a node to learn 75% of the network.

From this data we find that the metadata required increases when the path size limit becomes bigger, however this increase diminishes as the path size limit becomes larger. This is because most received messages have only traveled a limited number of hops, making the probability of a received message reaching the path limit lower as the limit increases. Similarly, the learning time required to learn three quarters of the network deviates largely when increasing a small path size limit, and barely deviates when comparing high path size limits.

The average completeness of a node changes very little when limiting the path size. However, when severely limiting the number of packets that will be analyzed this will have a large effect on the average completeness of the network. When looking at path sizes lower than 5 we find nodes that have a completeness lower than 75% of the entire network.

7. CONCLUSION AND FUTURE WORK

Path-based topology discovery has a high information gain in early stages of a network. But as the amount of received messages increases, the topological information gain of future messages decreases. At a certain point, the burden of having to

share the paths is larger than the information gain. When this happens is dependent on the network: the network size and the average amount of neighbors are likely factors that influence this. For future work it would be interesting to find out when this point occurs and in what ways certain factors influence it.

Also, those paths that do give new information about the network are likely to contain many edges that are already captured in the network map. Edges that provide the best connection for a node to a large part of the network will be seen often in these paths. Possible future work would be to detect the occurrence of edges and use this information to determine the value of an edge.

The number of neighbors a node has, has very little influence on the speed at which the node learns the network. Limiting the sizes of paths in the metadata of packets does have a large effect on the learning speed, decreasing this limit decreases the learning speed. Additionally, the overhead also decreases, whilst the average completeness of the network map for all nodes stays the same. This makes limiting the path size in headers a trade-off between learning speed and the overhead. The point at which the burden of the overhead outperforms the rate of information gain by limiting path size will be different for each network. Currently only one network was analyzed using the developed software program, for future work this software would be used on many networks.

To summarize, path-based topology discovery can be used to successfully map a network. Especially in the early stages of network discovery the algorithm is very efficient in learning the topology. When a network is already largely mapped the burden of all the required metadata is too high to justify the low information gain. Limiting the path size limit also helps with making the method more viable, as it allows learning speed to be sacrificed for lowered overhead costs.

8. REFERENCES

[1] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri and A. Silberschatz, "Topology discovery in heterogeneous IP networks," Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064), Tel Aviv, Israel, 2000, pp. 265-274 vol.1. DOI= https://doi.org/10.1109/INFCOM.2000.832196

[2] B. Donnet and T. Friedman, "Internet topology discovery: a survey," in IEEE Communications Surveys & Tutorials, vol. 9, no. 4, pp. 56-69, Fourth Quarter 2007. DOI= https://doi.org/10.1109/COMST.2007.4444750

[3] Diaz C., Murdoch S.J., Troncoso C. (2010) Impact of Network Topology on Anonymity and Overhead in Low-Latency Anonymity Networks. In: Atallah M.J., Hopper N.J. (eds) Privacy Enhancing Technologies. PETS 2010. Lecture Notes in Computer Science, vol 6205. Springer, Berlin, Heidelberg. DOI=https://doi.org/10.1007/978-3-642-14527-8_11