Distributed Routing Algorithms for Dynamic Geospatial Networks

Alex Tamkin, Cooper de Nicola, Raymond Yao, George Hosono

June 4, 2022

Abstract

Typical models of computer networks assume a mostly-fixed topology, where the underlying connectivity of the nodes is largely unchanged, and routing tables can be reasonably maintained. However, the advent of short-range wireless technologies and internet-of-things devices has led to the emergence of dynamic networks, where the network topology changes frequently enough that maintaining routing tables is infeasible. We consider several possible mathematical models of dynamic network topologies, propose several routing algorithms for efficient message delivery, and empirically evaluate and visualize these algorithms on a custom-built simulator. *(Final project for Stanford's CS 244B.)*

1 Introduction

The rises of small consumer devices and short-range wireless technologies has created a preponderance of *dynamic geospatial networks*: networks whose connectivity patterns depend on shifting distributions of agents in physical space.

Although they may seem quite similar in spirit to traditional computer networks, several key differences emerge when one wishes to send messages across such a network. Most computer networks, for example, have nodes which are typically fixed on server racks and whose connectivity patterns are largely unchanged. Because this network topology is largely *static*, nodes in the network can afford to maintain constant routing tables by computing expensive spanning trees which rarely need to be updated. These routing tables then enable each node to forward an incoming packet along to the next node on the path to its destination, according to the routing table.

In dynamic networks, however, the network topology changes frequently enough that one cannot hope to maintain routing tables due to rapidly shifting connections. Indeed, there is no guarantee that a shifting network topology will always have one single connected component. This necessitates fundamentally different algorithms which are resilient to, or even perhaps leverage, the shifting connectivity patterns of dynamic networks.

We present a custom-built network simulator to design, benchmark, and visualize routing algorithms for dynamic networks. Of particular interest are dynamic *geospatial* networks, whose network topology is dependent on the proximity of agents moving in physical space. We consider several distributed routing algorithms and evaluate and visualize them on two different dynamic network topologies, one of which is geospatial. We find several interesting properties of such algorithms, and we hope our insights are useful for developers of such algorithms.

2 Simulator

We opted for a highly flexible, robust design for the routing network simulator. Our aim was to easily enable for the (1) development of different forwarding algorithms of packets for individual nodes and (2) creation of many topologies (i.e. arrangements of connections between nodes). We draw heavily from object-oriented programming principles (abstraction and inheritance) in the design and implementation of our simulator.

2.1 Design

Our in-house network simulator is composed of the following six classes:

`NetworkSimulator`: Orchestrates the simulation of a network with a changing topology. Accepts a `Workload` and runs the `Workload` according to a `Node` packet forwarding algorithm and `Topology` class.

`Workload`: Consists of the `Message` objects that need to be routed from their start node to end node destinations.

`Message`: The raw information that needs to be passed from the start node to the end node. Currently, these messages themselves contain no real valuable information (as they only contain routing details and cost metrics), although information can easily be integrated by adding additional instance variables of any type to this class.

`Packet`: The information that is transmitted between nodes in the network. Wraps around a `Message` object and contains information about visited nodes. The logic behind divorcing a `Message` and `Packet` is to enable sending multiple packets *of the same message* to different nodes in dynamic networks.

`Node`: Handles receiving and sending of messages for individual nodes. This class contains the different packet-forwarding algorithms (such as breadth first search, breadth first search with early/ late splits) that we wish to explore.

Within our `Node` class, we implement a creative solution to segregate the `Packet` objects that are queued (and ready to propagate to another `Node`) — dubbed the *inbox* — and the `Packet` objects that are stored in the `outbox` to be processed on the next time step. This results in a system that can successfully distinguish between packets sent on the same timestep and intended to be sent on subsequent timesteps (in order to prevent packets delivered to another node to be processed on that same timestep).

`Topology`: The arrangement of nodes (i.e. connections between nodes) within the graph network. This class takes into account how connected we wish the network to be (*density*), as well as the frequency of changes within the network after each time step (*volatility*).

2.2 Metrics

We track numerous metrics to gauge the overall effectiveness of the routing algorithms. The metrics are easily configurable using command-line flags. They are as follows:

- **Total workload cost**: The cumulative number of "hops" for all messages in a workload to be delivered
- **Average packets per message**: The cumulative mean number of packets sent per message
- **Peak packets per node**: The maximum number of packets sent across all nodes in a workload
- **Average inbox load**: The cumulative mean number of packets in the inbox of each node

3 Topologies

Routing algorithms may be highly dependent on the network topologies they serve. To explore the effect of different network topologies on our routing algorithms, we consider two different dynamic network topologies which evolve over time. Each topology is parameterized by two values. The *density* $\rho \in [0, 1]$ controls how connected the graph is (with 0 meaning no connections and 1 being fully connected) and the *volatility* $\sigma \in [0, 1]$ controls how much the connections in the graph change each timestep. As we will see, the same algorithm can have markedly different properties across topologies.

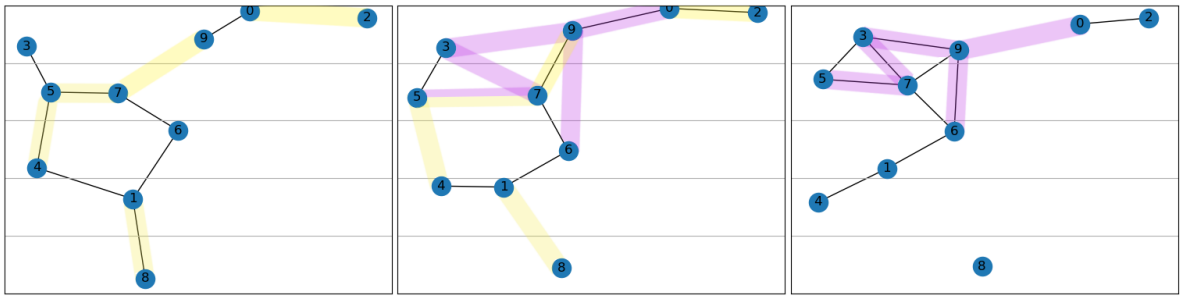


Figure 1: Three timesteps of the random geographic network topology changing. This network has 10 nodes, a density of 0.2, and a volatility of 0.07. Topological changes between timestep 1 (left) and 2 (middle) are in yellow; topological changes between timestep 2 and timestep 3 (right) are highlighted in purple.

3.1 Dynamic Unstructured Topology

Our first topology is inspired by the Erdős–Rényi model of random graphs [Erd59]. Here, every node is connected to each node with a fixed probability determined by density ρ. Thus, the network is entirely connected when $\rho = 1$ and entirely disconnected when $\rho = 0$. Volatility σ is introduced to the topology $\mathcal{T}$ each timestep by generating an independent alternate topology $\mathcal{T}'$ with the same ρ and mixing the topologies together: the connectivity between nodes n_1 and n_2 is determined by the respective connectivity in $\mathcal{T}$ with probability σ and $\mathcal{T}'$ otherwise.

3.2 Dynamic Geospatial Topology

The unstructured dynamic topology captures connectivity patterns where the connectivity status of two nodes is independent of all other pairs of nodes. However, in practice, the connectivity status of nodes may be dependent on external latent variables. For example, nodes in a network may be laid out in some geospatial arrangement, where their connectivity patterns are determined in part by their proximity.

To model some of this complexity, we consider a topology where nodes are placed uniformly randomly on a 2-dimensional unit square. If the Euclidean distance between two nodes is less than $\rho * \sqrt{2}$ then the two nodes are connected, otherwise they are disconnected.

Volatility is introduced to the network at each timestep by generating an alternate topology $\mathcal{T}'$ where the node locations are sampled independently. The node location in the new topology is determined by taking the old location of the node l_n and adding the L_2-normalized vector of the alternate location l' scaled by network volatility σ, such that $l_{n+1} = l_n + \sigma \frac{l'}{||l'||_2}$, followed by a slight adjustment to return any node escaping the simulation boundaries by reflecting them over the relevant boarder axis. Thus, high σ corresponds to a network where the nodes are moving faster relative to one another, and thus their topology is changing more rapidly. Figure 1 shows a geospatial topology graph varying over three timesteps.

4 Distributed Routing Algorithms

How can we deliver messages in a dynamic network? For static or near-static networks (where $\sigma \to 0$), typical approaches consist of generating a routing table for the network through an expensive search process that need to be performed very infrequently. However, in the case of dynamic networks ($\sigma > 0$), the network topology changes frequently enough that such an approach is computationally intractable. Instead, we consider the following distributed routing algorithms where nodes merely need to know their set of neighbors at any given time, but need not posses any global information about the network.

4.1 Random Forwarding

We first consider a random forwarding algorithm. Here, a node receives a number of message packets and randomly forwards one packet for each message to a single one of its neighbors. If the node has

no neighbors, it retains the message until the node gains a neighbor.

This algorithm is conceptually simple, and avoids overloading the network with too many packets. However, the disadvantage is that it can take many timesteps for packets to reach their destination.

4.2 Naive Breadth First Search

Another approach we consider is a breadth first search algorithm. Here, each node receives a message packet and sends a copy of that packet to each and all of its neighbors. Each node also keeps a list of which messages it has seen. If it has already seen a message, it does not forward it to other nodes.

This algorithm is conceptually simple, and minimizes the number of hops it will take for a message to reach its destination. However, the exponential growth of the packets can lead to congestion in the network. Furthermore, there is no way to know that packets have reached their destination. Copies of packets continue to be routed through out the network even after a message has been delivered.

4.3 Breadth First Search with TTL

This algorithm builds on Naive Breadth First Search by adding a "Time to Live" or "TTL" parameter to each packet. Whenever a packet is forwarded, the TTL is decremented by 1. When the TTL reaches 0, the packet is dropped from the network. This prevents a packet from being routed indefinitely. Below, we discuss the choice of TTL and how it impacts performance.

4.4 Breadth First Search with Early/Late Split

These two algorithms attempt to optimize Breadth First Search with TTL by duplicating packets less often. In Breadth First Search with Early Split, each node will send each packet it receives to a fraction of its neighbors, equal to the current TTL of the packet divided by the starting TTL of packets. In other words, packets will be duplicated more often towards the start of their journey (when TTL is high) and less often towards the end (when TTL is low).

In contrast, in Breadth First Search with Late Split, each node sends each packet it receives to a fraction of its neighbors equal to the starting TTL divided by the current TTL of the packet (with a minimum of one neighbor). This means that packets are rarely duplicated at the start of their journey (when their TTL is high) and are frequently duplicated at the end of their journey (when their TTL is low).

4.5 Naive Breadth First Search with Looping

Our original Naive Breadth First Search contains an optimization where nodes will not forward on packets containing message that they have already seen. This becomes an issue when a message's start and destination nodes are in different connected components of the network. Random Forwarding can forward a packet within one connected component until a change in network state allows the message to reach new nodes. In contrasts, Naive Breadth First Search drops packets before they can escape a single connected component.

Naive Breadth First Search with Looping works like Naive Breadth First Search, but it allows message to be forwarded through the same node multiple times.

5 Results

We plotted heatmaps showing how a given metric varies with the density of a network (i.e. the probability that any two nodes are connected) and the volatility of a network (a parameter indicating how often nodes move in the network). The heatmaps themselves are attached in Appendix A.

5.1 Random Forwarding

Random forwarding is an surprisingly powerful method of routing. It manages to successfully deliver more than 80 percent of messages in under 250 hops in unstructured graphs with a density above 0.18. It also performed well in geospatial graphs successfully delivering more than 80 percent of messages in under 250 hops in unstructured graphs with a density above 0.3. In geospatial graphs with a volatility

of 0.1, Random Forwarding successfully delivered more than 80 percent of messages when the density was above 0.1.

The main drawback of Random Forwarding is the large number of hops it takes for a packet to be delivered. In low density networks, packets are forwarded randomly and are often never delivered. In high density networks, packets may take circuitous routes to their destinations. For instance, in a complete graph (density=1) of 100 nodes, Random Forwarding takes, on average, 50 hops for a message to be delivered, even though the message can be delivered in one hop.

See Figure 2 for more details.

5.2 Naive Breadth First Search

Naive Breadth First Search is able to deliver packets more quickly than Random Forwarding but at the cost of more packet duplication. It manages to deliver 100 percent of packets in geospatial networks with a density above 0.18. However Naive Breadth First Search often creates more than 90 packets for each message in networks with 50 nodes.

Naive Breadth First Search often fails to deliver messages in networks with density below 0.18. This is because our implementation forbids nodes from forwarding on packets they have previously seen. This means that if the source and destination of a packet are in different connected components of the network, the packet will reach every node in that connected component and then be dropped.

See Figure 3 for more details.

5.3 Breadth First Search with TTL

Breadth First Search with TTL is able to deliver messages at almost exactly the same rate as Naive Breadth First Search. Additionally, it did not significantly reduce the number of packets created per message. Our initial TTL was set to the number of nodes in the graph. It seems that this was high enough that packets were dropped by being routed in a circle before they could be dropped due to their TTL.

See Figure 4 for more details.

5.4 Breadth First Search with Early Split

Breadth First Search with Early split performs similarly to Naive Breadth First Search. Packets are usually dropped for having been routed in a cycle before they stop splitting.

See Figure 5 for more details.

5.5 Breadth First Search with Late Split

Breadth First Search with Late Split performs similarly to Random Forwarding. However, this algorithm does not allow packets to be forwarded back to a node that has already seen a packet, so it is less likely to successfully deliver packets. Our parameters for splitting means that before packets can split, they are usually dropped for having been routed in a cycle.

See Figure 6 for more details.

5.6 Naive Breadth First Search with Looping

Naive Breadth First Search with Looping performed remarkably similar to Naive Breadth First Search. The only difference between the two algorithms is that Naive Breadth First Search with Looping allows a node to route a packet containing a message even if it has seen that message before. This allows packets to travel in loops around connected components while it waits for a change in network state to open a path to the destination node.

Naive Breadth First Search with Looping successfully delivered 100 percent of messages in geospatial networks with a density of 0.1715. This is a slight improvement over Naive breadth first search which successfully delivered over 90 percent of messages at that density.

Naive Breadth First Search with Looping creates slightly more packets than Naive Breadth First Search. Naive Breadth First Search with Looping created more than 95 packets per message in networks with 50 nodes.

Overall, Naive Breadth First Search with Looping delivers messages more consistently than Naive Breadth First Search with a small amount of additional overhead.

Both algorithms perform poorly with densities less than 0.1715. In this case, there is frequently no route from a source node to its destination, so it is impossible to deliver messages.

See Figure 7 for more details.

5.7 Problems with Disconnected Networks

In low densities networks, nodes are often partitioned among different connected components. If a source and destination node are not in the same connected component, then there will be no way to route a message between them. This problem is especially bad for breadth first search algorithms. Those algorithms flood their connected component with traffic that never reaches its destination.

We suggest that future research separate the problem of routing packets into two different problems: identifying if there exists a path from the source to the destination and routing packets when such a path exists. If there exists no path between a source and destination, then nodes can either drop a packet or hold on to it until some change in network state opens up the possibility of delivering the packet.

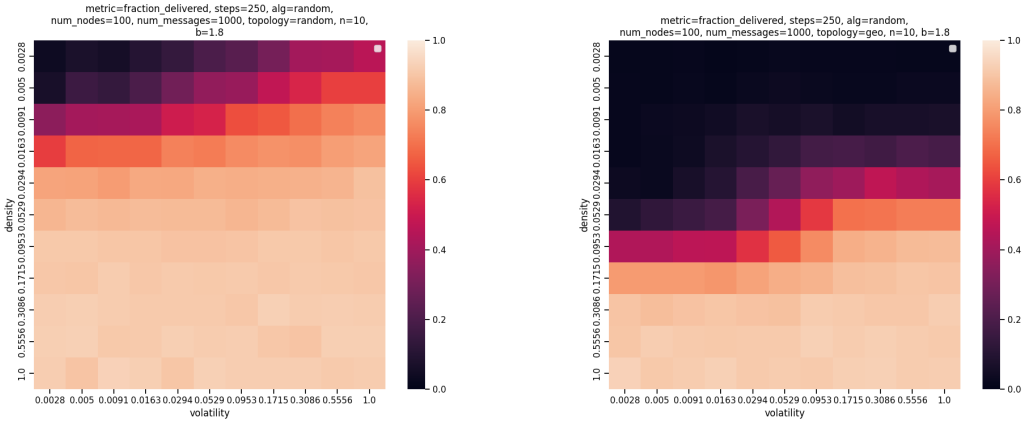
6 Conclusion

We explore several distributed routing algorithms on dynamic network topologies. Our results show that random forwarding is a simple yet performant algorithm that works well across a range of network densities and volatilities. Furthermore, broadcasting-based approaches such as the variants of BFS we tried can lead to faster message delivery at the cost of congestion in the network.

References

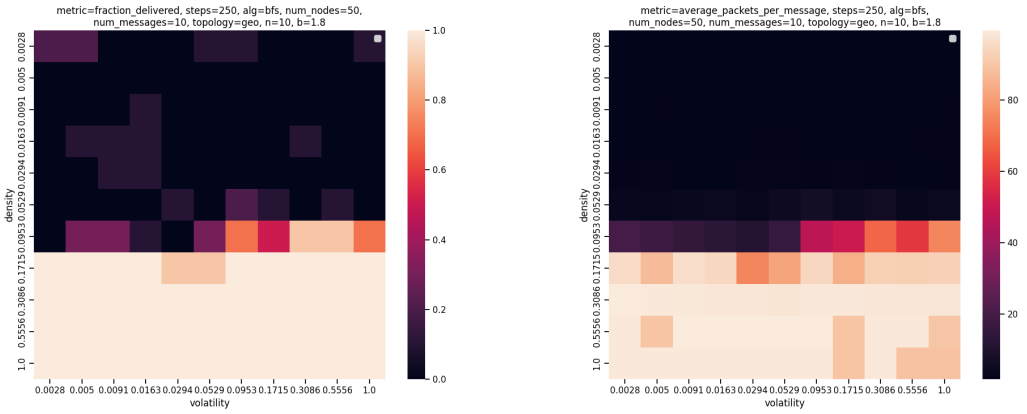
[Erd59] Paul Erdös. On random graphs, i. 1959.

Appendix A: Figures



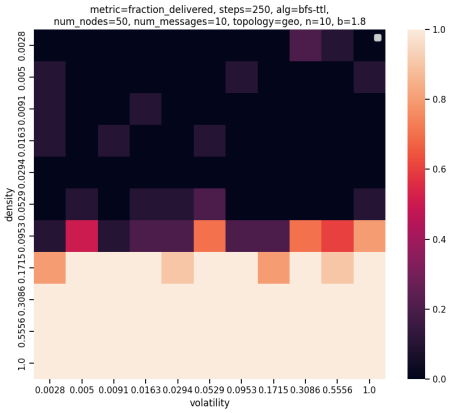
(a) Unstructured Topology (b) Geospatial Topology

Figure 2: Results of random forwarding algorithm on both topologies

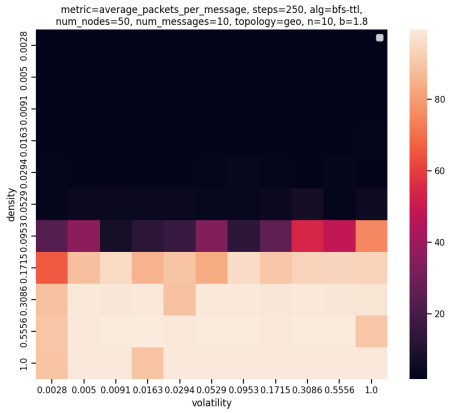


(a) Fraction of Messages Delivered (b) Packets Created per Message

Figure 3: Results of Naive Breadth First Search

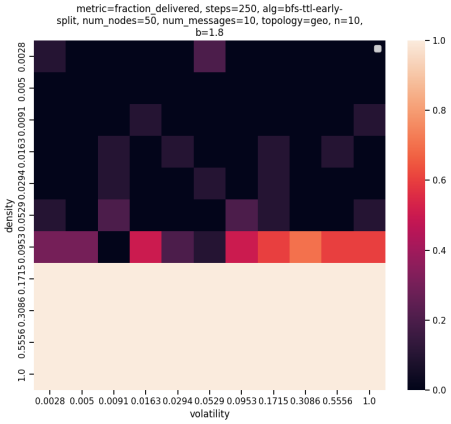


(a) Fraction of Messages Delivered

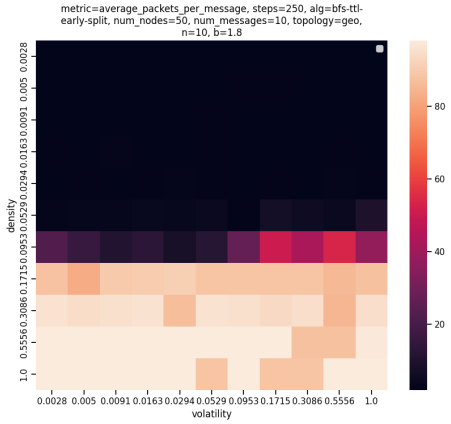


(b) Packets Created per Message

Figure 4: Results of Breadth First Search with TTL

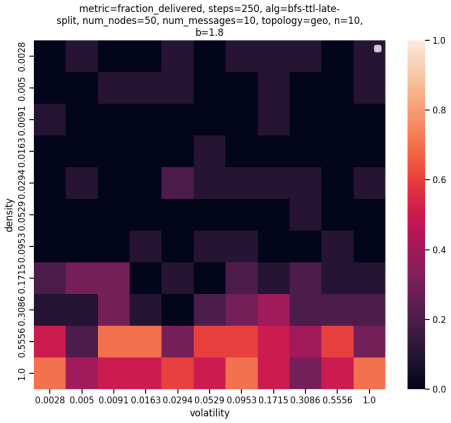


(a) Fraction of Messages Delivered

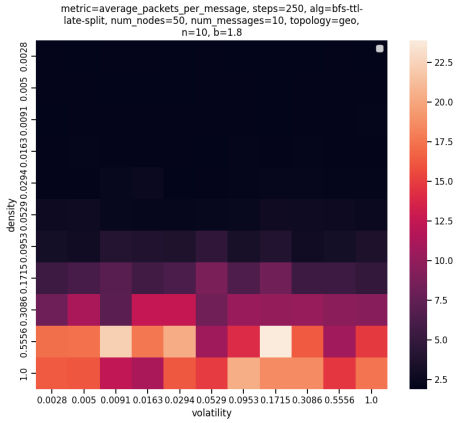


(b) Packets Created per Message

Figure 5: Results of Breadth First Search with Early Split

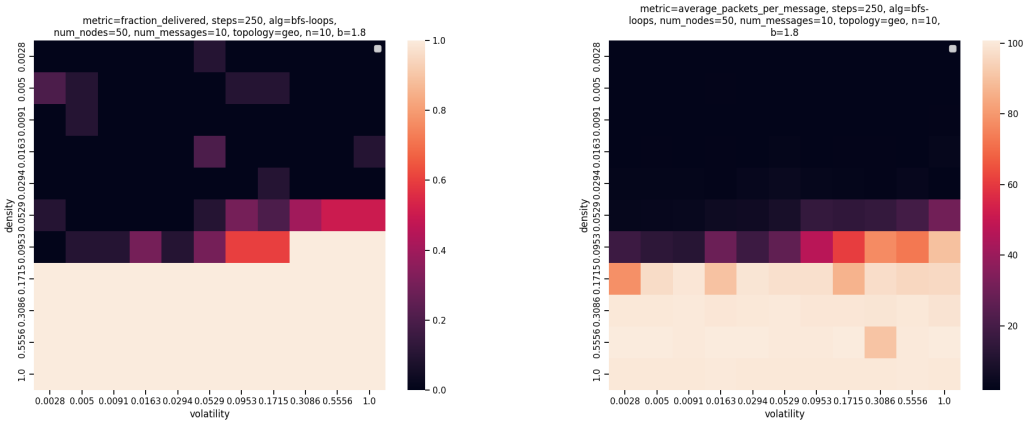


(a) Fraction of Messages Delivered



(b) Packets Created per Message

Figure 6: Results of Breadth First Search with Late Split



(a) Fraction of Messages Delivered

(b) Packets Created per Message

Figure 7: Results of Naive Breadth First Search with Looping