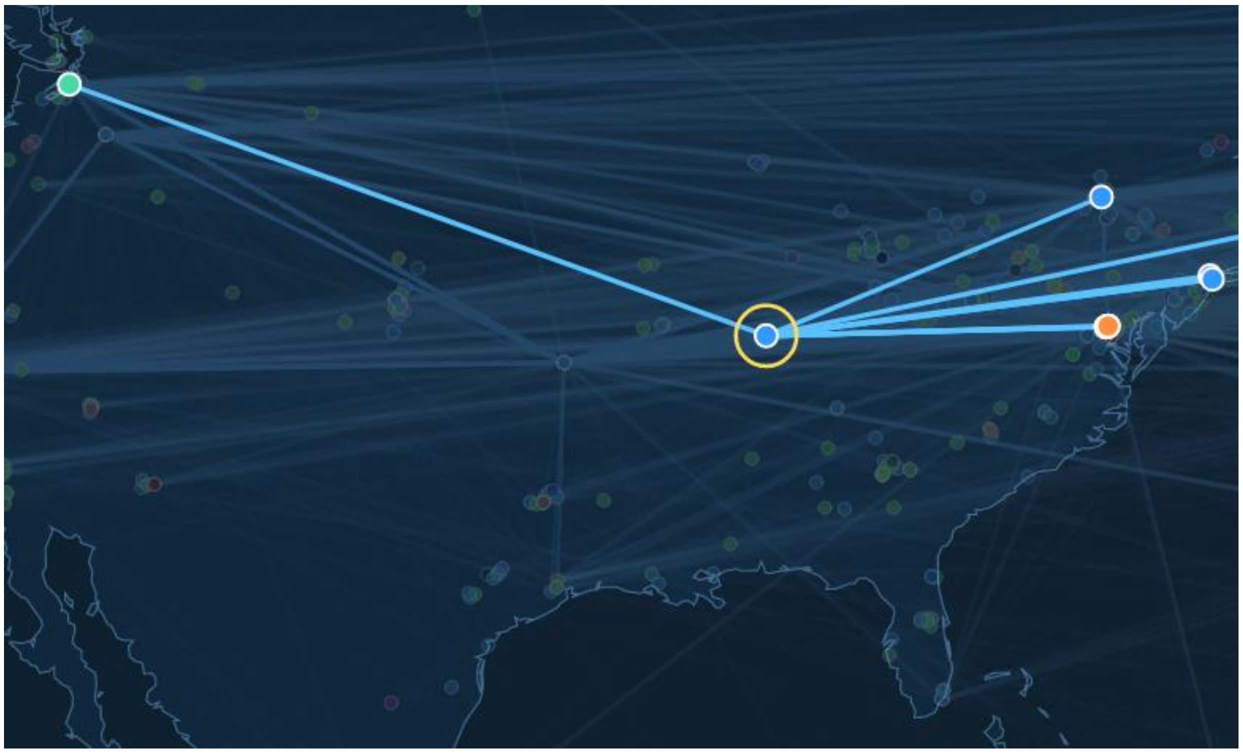
A Lightning Network Modeling and Simulation Framework

An Open-Source Development Proposal

The Lightning Network is a new and exciting technology that will likely change how Bitcoin is used all over the world. It is a complex, open network that is constantly changing and evolving. It is hard to predict its behavior due to the decentralized nature of it and as more and more people start using this network the behavior will become more advanced and even more unpredictable.

Throughout history as new technologies are developed and complex systems are built using those technologies people have built Modeling and Simulation tools alongside of them in order to better understand them and to test them. The same should be done for the Bitcoin Lightning Network.

A tool is needed that allows developers and researchers to define a Lighting Network layout and then simulate events over time to analyze how the network might be impacted. This could be used to test new products that a team is developing or to help research and predict how certain events might impact a Lightning Node or the Lightning Network as a whole.


How does the operation of the Lightning Network change if the selected node goes offline for 2 days?

For example, let's say a large routing node that handles lots of transactions and has lots of open channels goes offline for an hour. This has the potential to have a big impact on the network:

- How would this event impact transaction fees during the outage?
- Which nodes would be most affected?
- How many failed transactions might occur during this outage?
- How would all these statistics be different if that outage was 30 minutes instead of an hour? What about a day instead of an hour?

All these things could be simulated and then analyzed. Charts and reports could be generated so that the user can make decisions based on this data.

The Problem

In order to develop new products and services using the Lightning Network, engineers need a quick and easy way to spin up a complete Lightning Network to test their work. This network includes Lightning Network nodes, Bitcoin nodes, transaction histories, channels with different liquidities, etc. Before connecting a new service to the testnet or mainnet often a local "regtest" network is used.

There are lots of tools out there to help create regtest networks, but these tools lack two things:

1. The ability to automate different events over a period of time and gather data during those events. As time goes on nodes might go offline, close channels or run out of liquidity. All these events will change how the network operates. Current tools require the user to manually interact with the nodes on the network in order to generate these kinds of events.
2. The ability to model a large network. Current tools typically use docker containers for each lightning node and this approach works well for smaller networks. However, for larger networks docker containers will typically be too resource intensive to allow for large scale networks to be modeled.

Projects like "Polar" allow users to design a test network and manually interact with it by doing things like creating an invoice, opening a channel, etc. A lot of these projects use full Lightning Network node implementations and/or docker containers to set up a realistic network. What if a user wanted to set up a network with a thousand nodes and have those nodes pay each other several times over the course of an hour? This would not be possible with the current tools because it would be difficult to run a thousand docker containers and manually create and pay invoices repeatedly. In addition, these tools do not have analytics built into them to report on the status of the network as these events take place.

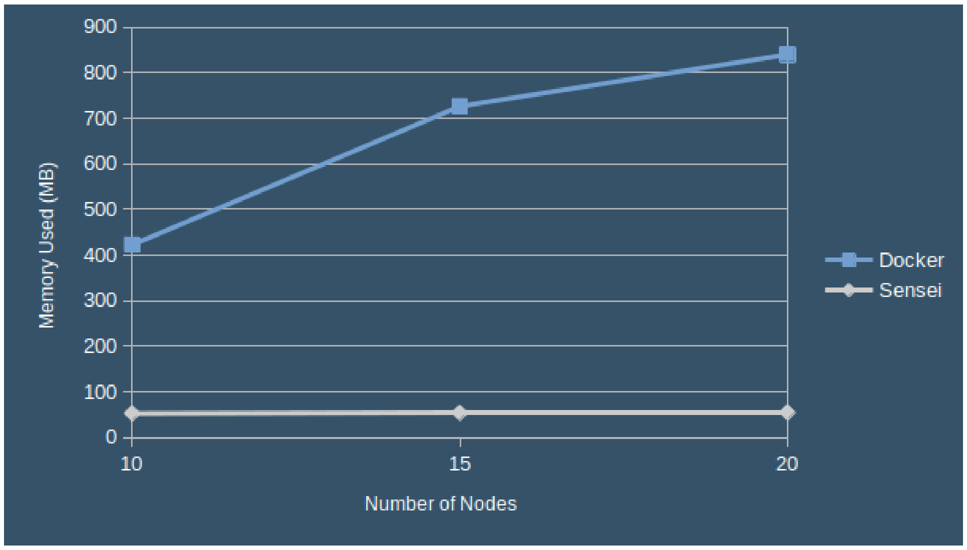
The Solution

A solution could be built as an open-source library that models the behavior of the Lightning Network. This library would allow a developer to specify characteristics of the network and all its nodes. As well as defining the initial state of the network, a user could define events that will take place sometime in the future. Then by running the simulation the user could see how those events impact the network over the specified time period.

A user could define all the nodes in the network and choose which ones are connected to each other. Along with the number of nodes and connections, a user could define events that take place at different timestamps, for example:

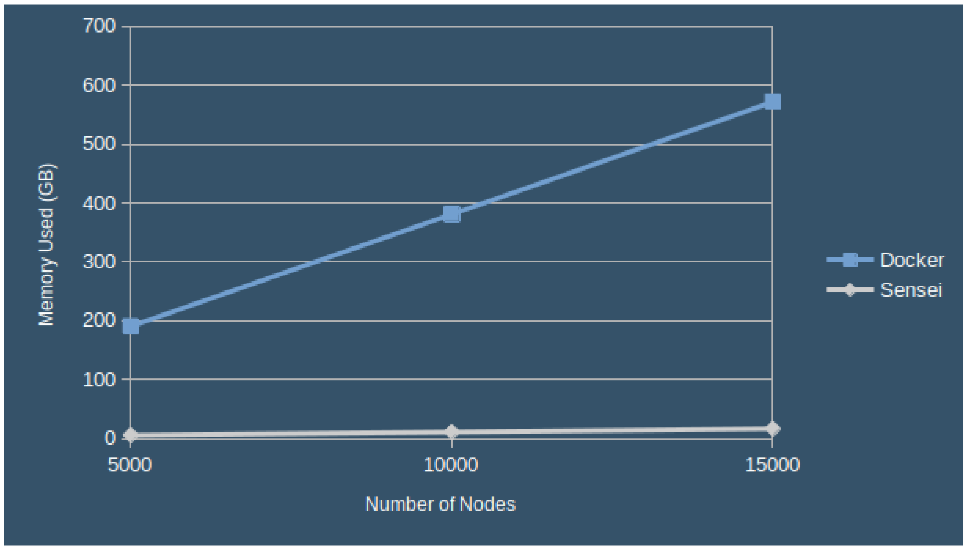
- A node closes all its channels
- A node opens new channels
- A node raises its routing fees
- A node goes offline

Projects like “Sensei” or the LDK itself let users set up "lightweight" Lightning Network nodes. These nodes share resources in a way that makes them very efficient. This is the perfect implementation to use as the core of a simulation tool because these “lightweight” nodes would allow for large networks to be modeled. Here is a comparison of two Lightning Networks, one created with docker containers, and one created with Sensei.



This graph shows the amount of host memory used (in MB) for a network of 10, 15, and 20 nodes. Note that these networks do not have any channels, they are simply made up of Lightning Network nodes that are running and waiting for connections. This shows how adding nodes to the docker network will increase the memory needed much faster than adding nodes to the Sensei network.

In this first scenario it was seen that each docker container used on average about 38 MB of memory. This is not an issue for a network of 20 nodes, but what happens when you need a network closer to the actual size of the Lightning Network mainnet? Here is a projection of how much memory could potentially be needed.



This graph shows the amount of host memory used (in GB) for a network of 5000, 10000, and 15000. Sensei can easily handle these large networks and still provides a realistic model of each Lightning Network node and for those reasons it is a great candidate for the Lightning Network implementation used in a simulation environment.

The Project

A back-end library could be built so that users could quickly write extensions/plugins that would create different types of networks with the user's desired qualities and events. A front-end UI could also be built so that a user could visualize a network while they design it for their simulations and watch different nodes and the network as the simulation runs. Along with the simulated network an analysis/reporting tool could be built to report network statistics, events, and the overall state of the network. This would allow developers to see how their products are impacting the network as time passes and as different events take place. The proposed project would be divided into two main parts:

1. **ln_ms_lib** - a Rust library that exposes an API that will let developers quickly create and run simulations. Part of this library would be a way to parse network definitions from several different sources. For example, projects like Polar... where a developer has defined a test network using Polar and has been testing their app. Polar could integrate this library and the user could quickly turn their test network into a simulation, define their events and run it.

2. **ln_ms_server** - a Rust back-end web server that exposes the ln_ms_lib library API to a web client. This could be used to build a web front end that would serve as a standalone simulation tool where a user could define their network through the UI and then run simulations.

By keeping the simulation library outside of any other projects and keeping it lightweight and flexible, it could serve several different purposes and be used differently by different projects.

Proof-Of-Concept GitHub Repository

https://github.com/bjohnson5/ln-ms-framework

About Me

Name: Blake Johnson

Email: brjohnson5@icloud.com

LinkedIn: https://www.linkedin.com/in/blake-johnson-11aa05206

I am an experienced Software Engineer who has worked on several different Modeling and Simulation projects in my career. Most of my experience has been in the defense industry building simulations and models for the military. The defense industry has seen firsthand the benefits of high-fidelity simulation products that accurately represent real world systems. New products and features can be tested in a closed, simulated environment before being tested in the real world (which can be expensive and inefficient). The same benefits could be realized by the bitcoin development community. A large, flexible, open-source simulation framework would allow the Lightning and Bitcoin communities to more efficiently test and release new services. It could also help create stability in the network over the long term.