A Lightning Network Modeling and Simulation Framework

An Open-Source Development Proposal

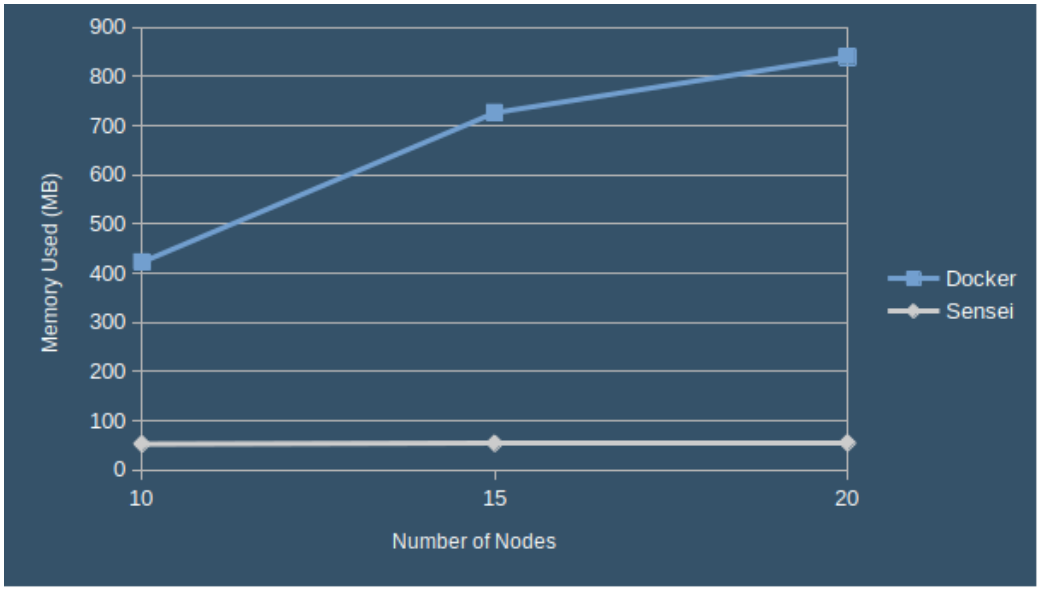
Bottom Line Up Front: I have created a minimum viable product for a robust modeling and simulation library for the Lightning Network. I would like to expand this library into a world class simulation tool based on my 8 years of experience building M&S products for the defense industry. You can find a proof-of-concept for the project here:

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

Current regtest and simulation environments such as Polar use a Docker infrastructure that cannot scale to model the behavior of the wider Lightning Network. These tools work well for small scale users when testing how a few nodes will interact, but they cannot simulate the broader network as required by a large LSP or business. I would like to build out this project as a best-in-class network simulation tool for use by companies/open-source projects that need to create large test networks.

A new project based on the LDK called Sensei allows users to set up lightweight Lightning Network nodes. These nodes share resources in a way that makes them very efficient, while still operating like a full Lightning Node. This is the perfect implementation to use as the core of the simulation product because these lightweight nodes allow for large networks to be modeled.

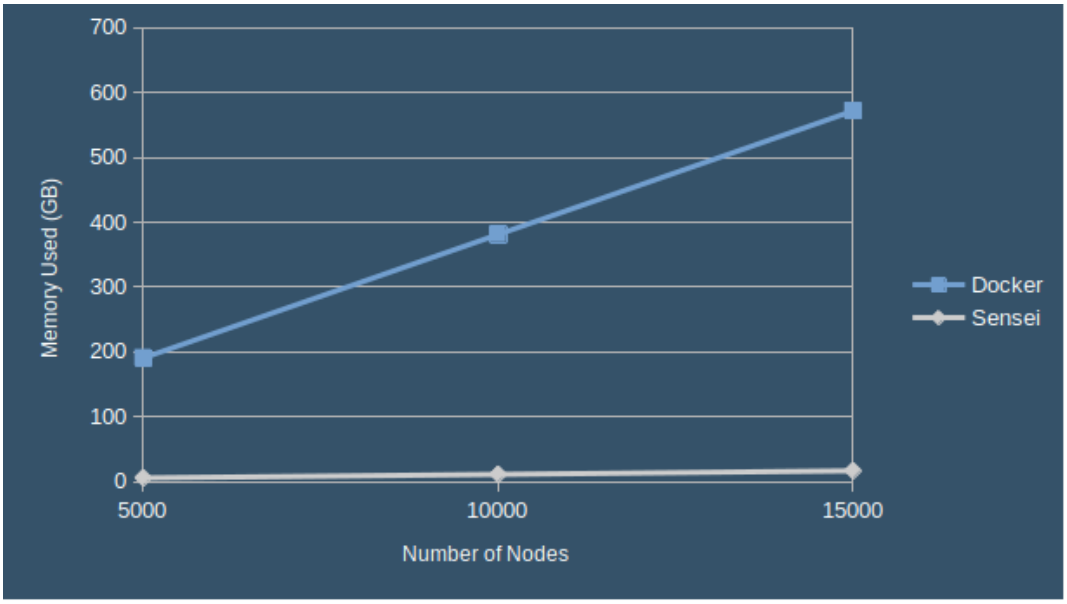
Here is a graph showing the amount of memory used during some initial testing. As you can see, my approach of using Sensei nodes allows for modeling much larger networks.



This new simulation framework leverages the flexibility of the LDK and Sensei to create a powerful simulation tool that will set the standard for how Lightning Network testing and research is done.

The previous 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 a Docker network will quickly become unrealistic as a solution for modeling large networks.

In the 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.

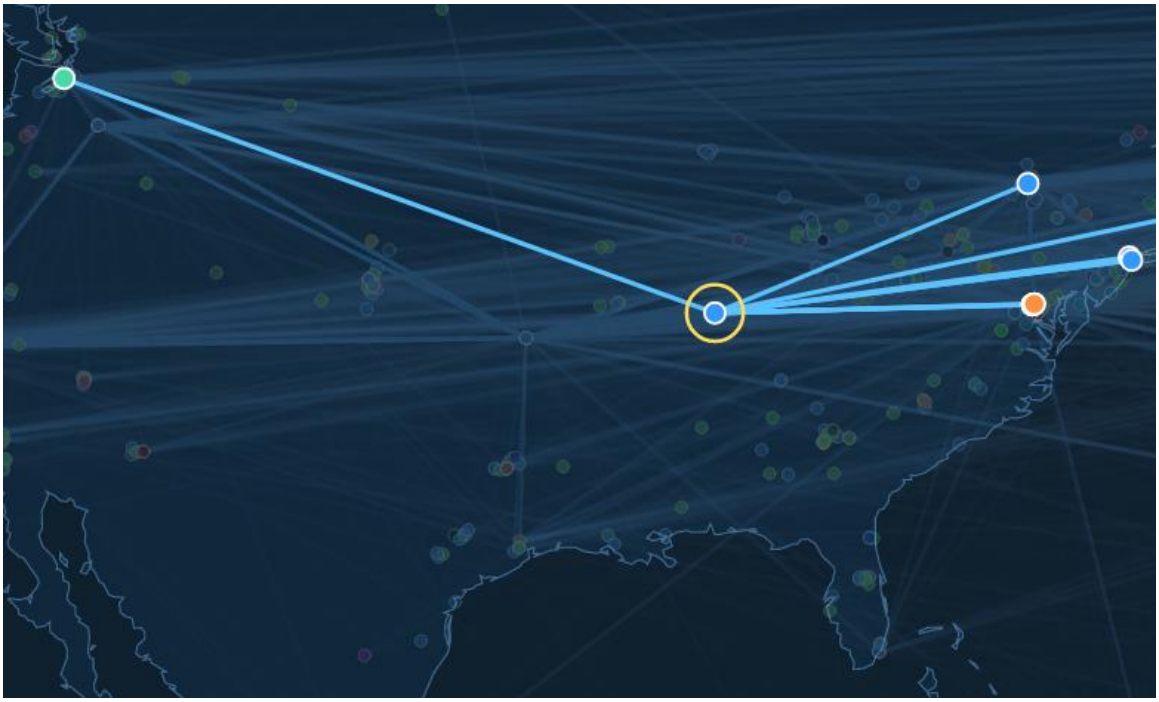


This graph shows the amount of host memory used (in GB) for a network of 5000, 10000, and 15000. Sensei can easily handle these large networks and still provides a realistic model of each Lightning Network node and for those reasons it is a great candidate for the Lightning Network implementation used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

Another thing that the current tooling lacks is the ability to automate different events over a period of time and gather data during that time. 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. What if a user wanted to set up a network with a thousand nodes and have those nodes create invoices, connect to peers, open channels and make payments? 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. Users should be able to collect data on the network and review that data as the simulation runs and after it is finished in order to observe how the nodes and network reacted to different conditions.

Lightning developers and researchers need a tool that allows them to define large Lighting Networks and then automatically simulate events to analyze how the network might be impacted by certain conditions. With my tool developers can test new products or research and attempt to predict how certain events might impact a Lightning Node or the Lightning Network as a whole.



How does closing a channel to this peer affect my payment success rate?

For example, let's say a large routing node that handles lots of transactions and has lots of open channels runs out of liquidity on a major channel. This has the potential to have a big impact on the network:

- How would this event impact transaction fees?
- Which nodes would be most affected?
- How many failed transactions might occur?

An LSP or routing node can use this simulation tool to set up different scenarios and test how different liquidity distributions could improve the profitability or stability of their node. Companies or teams that are building custom software to manage routing nodes can use the simulation tool to see how their tools react to certain situations that occur on the network.

The data generated from simulations can be used to compare results with other simulation runs, to research theories, and to see trends/patterns. This will dramatically change how testing is done and will set an industry standard that can be used by a variety of members throughout the bitcoin development community.

Because the nodes are based on the LDK, the simulation is very accurate at the protocol level and gives users the ability to connect their own nodes (running in regtest mode) to the simulated network and communicate reliably with the LDK nodes. This gives users confidence that the behavior of the simulation is accurate and well tested.

However, as with any simulation software, certain assumptions will have to be made. These assumptions include the total liquidity of the network, the channel layout and the amount of inbound and outbound liquidity that pre-configured nodes will have. Another assumption is that only very basic channel liquidity management will be done on the nodes in the simulation. Each of these assumptions can be configured by the user before running the simulation. Giving the user as many options as possible allows them to make their own assumptions about the state of the network for their simulation.

The proposed simulation framework gives users the ability to define nodes very precisely. The initial on-chain balance, channels, inbound/outbound liquidity, and up/down times are all examples of properties that can be set for each node in the simulation. Of course, if a user is setting up a network with thousands of nodes it is unrealistic that they will define every node this way. That is why the simulation tool allows a user to define how many pre-configured nodes to add to the network. For example, a user could define 10 nodes with their desired liquidity and settings and in addition tell the tool to start up 200 pre-configured nodes. Assumptions about these pre-configured nodes would have to be made, including the overall liquidity, channel connections, routing fees, etc...

Here is a simple code example of how the simulation library can be used to programmatically set up a large test network. This test simulation creates a network of 100 pre-configured nodes and 2 user-defined nodes. It opens a channel between the user-defined nodes and the user-defined nodes go online and offline during the simulation.

```
// Create a simulation that will run for 2 minutes and contains 100 pre-defined nodes
let sim_name: String = String::from("Test Simulation");
let mut sim: LnSimulation = LnSimulation::new(sim_name, dur: 120, num_nodes: 100);

// Create the user-defined nodes
let node_name_1: String = String::from("test_node_1");
let node_name_2: String = String::from("test_node_2");
sim.create_node(name: node_name_1.clone(), initial_balance: 500, running: false);
sim.create_node(name: node_name_2.clone(), initial_balance: 600, running: false);

// Create the user-defined channels
sim.create_channel(node1_name: node_name_1.clone(), node2_name: node_name_2.clone(), amount: 500);

// Create the user-defined events
sim.create_node_online_event(name: node_name_1.clone(), time: 30);
sim.create_node_online_event(name: node_name_2.clone(), time: 60);
sim.create_node_offline_event(name: node_name_1.clone(), time: 110);
sim.create_node_offline_event(name: node_name_2.clone(), time: 115);

// Run the simulation
let success: Result<(), Error> = sim.run();
assert_eq!(success.is_ok(), true);
```

One of the biggest assumptions that must be made about the network is the overall liquidity distribution of the network. The goal of this simulation framework is to give the user as much or as little control over this assumption as they would like to have. In order to do this, "node set profiles" can be created by the user to define the properties of a set of nodes in the simulation. Some users may want to specifically define each node in the network, but other users might want to create a large set of nodes with similar properties and only manually configure a few nodes. "Node set profiles" allow for multiple sets of nodes to be created with different profiles in order to simulate a large network with a variety of attributes. The 100 pre-defined nodes in the previous example will be run with a default “node set profile” but the user can also create their own profiles. Here is an example of two profiles that could be used to set up node sets.

```
{
    "initial_on_chain": "50000-100000",
    "number_channels": "1-10",
    "outbound_percentage": 100,
    "routing_fee": ""
}
```

senders.json

```
{
    "initial_on_chain": "50000-100000",
    "number_channels": "20-30",
    "outbound_percentage": 50,
    "routing_fee": "1-5"
}
```

routers.json

```
// Create a simulation and add 100 sender nodes and 500 router nodes
let mut ln_sim: LnSimulation = LnSimulation::new(name: String::from("test"), dur: 10, num_nodes: 0);
ln_sim.create_node_set(number_of_nodes: 1000, profile: String::from("senders"));
ln_sim.create_node_set(number_of_nodes: 500, profile: String::from("routers"));
```

The “senders” node set profile creates nodes with a small number of channels, each with 100% outbound liquidity. These nodes will start with a random balance between 50000 and 100000 sats and will use that balance to create a random number of channels between 1 and 10. These nodes will only be able to send payments because they do not have inbound liquidity. The “routers” node set profile creates nodes with more channels, each with 50% outbound liquidity and 50% inbound liquidity. These nodes will start with a random balance between 50000 and 100000 sats and will use that balance to create a random number of channels between 20 and 30. These nodes can send and receive, and each have a routing fee between 1 and 5 sats.

These profiles are user defined to allow the user to make their own assumptions about the overall liquidity of the network. They can set up as many or as few profiles as needed.

Another part of this library will be a way to parse network definitions from several different sources. For example, a user can export their test network from Polar and import it into the simulation tool and then run automated tests on it. Another example would be taking the network graph from a mainnet Lightning Node (using 'lncli describegraph') and importing that into the simulation tool.

During this network import, a user can specify what node profiles to use to fill in the information that is not publicly available in the import file. Importing network topology from the mainnet provides a way to easily create a large network that is accurate and gives the user a great starting point for building out a realistic test network. When importing a network file, an "import map" can be specified that will map public keys to node profiles. In this situation the node profile will only be used to set up the information on the node that is not available in the import file... for example, the inbound and outbound liquidity of each channel. Nodes from the import file that are not included in the import map will use the default node profile. This gives the user control over the assumptions that must be made about the network being imported.

Here is the default import map:

Additional features of the simulation framework include:

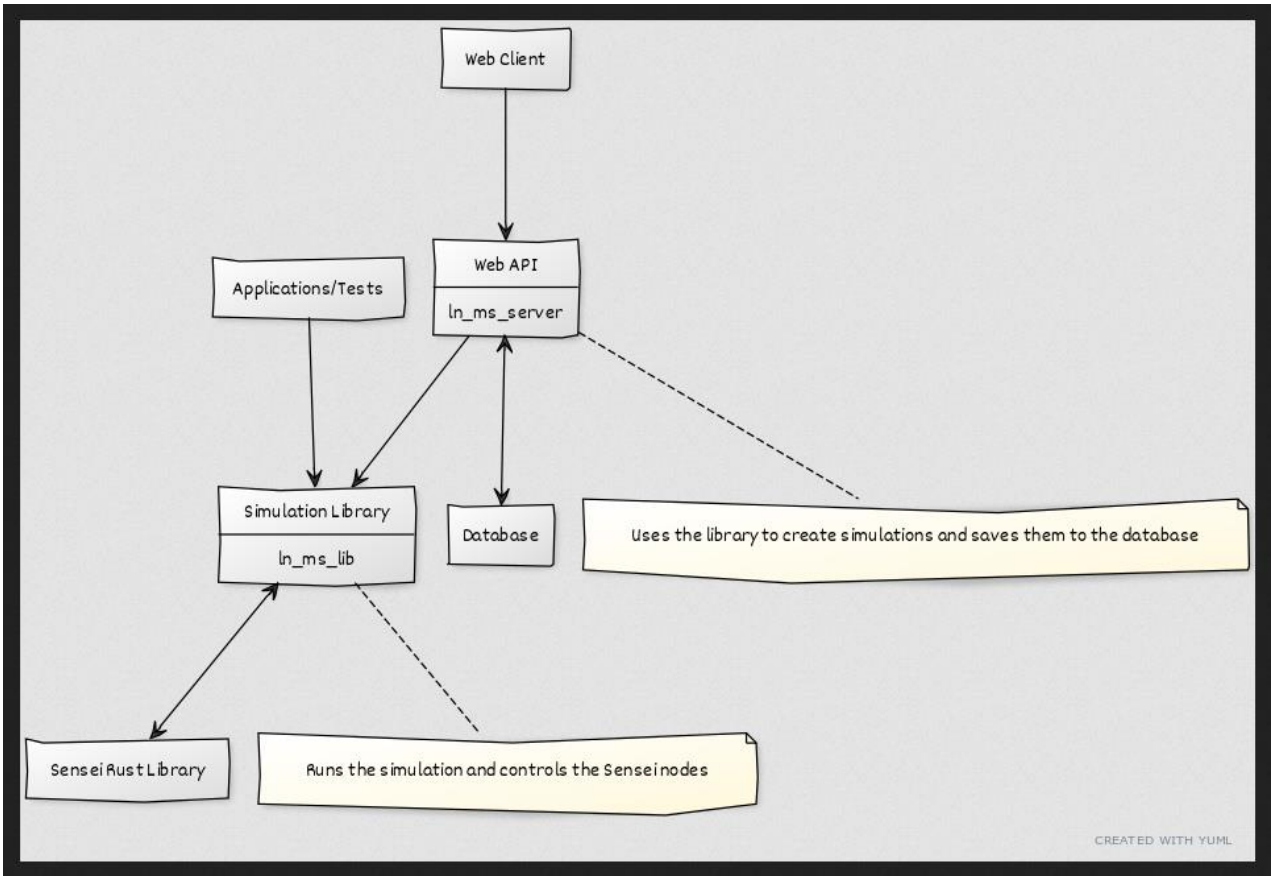
1. An analysis/reporting tool that runs alongside the simulated network to report network statistics, events, and the overall state of the network. Reports showing failed transactions, average fees, liquidity states for each node will be generated after the simulation so that the user can analyze the data and make decisions based on that data. The network analyzer will provide insight into the test network so that users can compare different runs, test routing strategies, or try to determine the overall reliability of the network.
2. A transaction generator that will simulate network traffic by creating and paying invoices. How often transactions are generated and between which nodes is defined by the user in a settings file.

Along with the core simulation framework a front-end UI will also be built so that a user can visualize a network while they design it. The UI will also be used to observe the network as the simulation runs. This would allow developers to see how their products are impacting the network as time passes and as different events take place. This will also allow them to save simulations to a database so that they can be re-run or edited after being created. By creating a shared library and a separate web server the project is very flexible and can be used to build on top of or as a standalone tool.

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. This would let developers design automated tests as they build new products.
2. **ln_ms_server** - a Rust back-end web server that exposes the ln_ms_lib library API to a web client. This will 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.

Here is a system diagram of the project showing how the tool can be used:



In summary, in order to develop new products and services for the Lightning Network, engineers need a quick and easy way to spin up a complete Lightning Network to test their products. There are lots of tools out there to help create regtest networks, but these tools lack two things:

1. The ability to model a large network. Current tools are too resource intensive to allow for large scale networks to be modeled.
2. The ability to automate different events and gather data during those events.

The proposed project will solve these issues by creating an accurate, scalable, and flexible modeling and simulation framework that can be used in a wide variety of ways. The project gives the user the ability to create large test networks by using lightweight LDK based nodes and provides powerful automation and reporting tools alongside those nodes.

Contact Me

Name: Blake Johnson

Email: brjohnson5@icloud.com

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