Methodology

To model a networked distributed decentralized system that utilizes a consensus protocol the key aspects of the system that we want to represent has to be defined. Those key aspects are:

- The performance time of the system to find consensus
- The resilience of the system to be able to function effectively
- These same effects of the system with varying network topologies

The methodology focuses on representing these key aspects and the effects they have on the total system. In order to better understand and test the performance and resilience of these systems, the consensus protocols need to be modeled at the node level. Modeling the system at the node level allows for the representation of the specific protocols reliance on number of nodes and how they operate within varying network topologies. To develop the model and simulation, a network simulator will be used with applications defined on nodes of the consensus system. Data will be observed and recorded from an ideal system that has minimum influence from external factors. This data will then be utilized in a way that it can be modeled for various components of the model development. Figure 1 shows the collection and use of the data from both the ideal system and the simulated system. Finally the developed model will be fine tuned and then validated to provide a meaningful measure on its effectiveness to represent the ideal system.

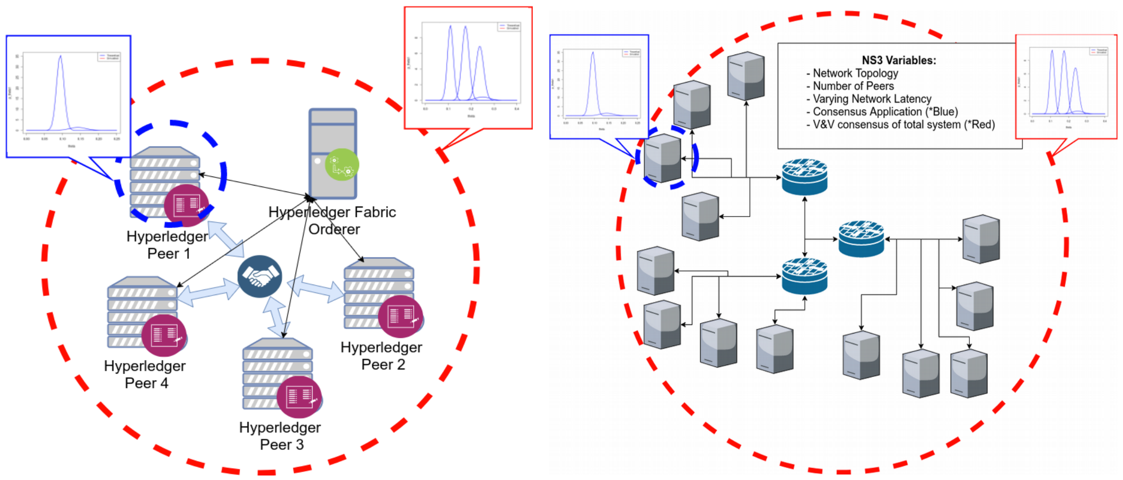

Figure 1: Demonstration of the data collection from the ideal system on the left and the application of the data to the simulated system on the right, blue indicating data gathered from individual nodes and red indicating data gathered from total system

Ideal System

An ideal system is a set up of the consensus protocols where external influences are minimized. The purpose of the ideal system is to provide a platform that can be easily modeled and one that would provide results as a baseline scenario. For this project the ideal system will be one where every node has a fast short network connection to each other in a very efficient mesh style topology. A good setup for this would be to utilize the virtual network on a single machine using docker containers. The nodes that exist on the ideal system is essentially just the participating nodes required to run the consensus protocol. The number of networked nodes will be kept at a minimum at first but will also be on the same setup with additional nodes in order to identify how performance changes as number of nodes scale. Logging must be enabled on the system so that data can be recorded on performance aspects of the consensus protocol.

Data Collection

The ideal system is the ground truth for this project and will be run with each consensus protocol that is to be modeled. Data will need to be collected from a couple different perspectives so that the key aspects can be represented in the modeled system. The data points to be recorded are listed below and details in recording these values will follow:

- *nMessage:* the distribution that represents the number of messages received, processed, and sent to accomplish consensus by each participating node that we will call
- *nodeDelay:* the distribution that represents the delay or time required for a participating node to process each message required to accomplish consensus.
- *nodeConsen:* the number of participating nodes required for the consensus protocol to find effectively find consensus
- *consenDelay:* the distribution that represents the delay or time required for the consensus protocol to reach consensus
- *consenState:* the probability that the consensus system will change to a special statewide
- *stateLength:* the distribution that represents the time the consensus system will remain in the special state

Node Messages

In the ideal system from the participating node the number messages will be recorded during the consensus process. The data *nMessage* will be recorded by counting the number of messages received and sent while consensus is being determined. The flow for recording this is as followed:

1. A transaction is proposed to the consensus system
2. from the participating node increment received counter when messages received
3. increment sent counter responding messages sent
4. when consensus is found report counter results and initialize them to zero

This process will be done many times in order to get a good representation of the number of messages a participating node must process during the process of finding consensus. The total dataset can then be processed and fitted to a distribution that represents the number of messages a participating node processes during consensus

Node Delay

The *nodeDelay* can be recorded the same time the *nodeMessage* is recorded. The participating nodes take a small amount of time to process each message during the consensus process. The data *nodeDelay* is essentially the time delay between each message received and each message sent during the consensus process. The flow for recording this is as followed:

1. A transaction is proposed to the consensus system
2. from the participating node when a message is received record a time stamp
3. when the participating node responds record a time stamp
4. initialize both time stamps
5. report received and sent timestamps
6. when consensus is found output time stamps for processing

Data will be gathered for a number of messages preferably simultaneously to the recording of *nodeMessage*. Once the data has been collected, the difference between the timestamps of the messages can be processed to derive the time it took the participating node to process each message. This data can then be used to fit a distribution that represents the typical time to process messages during consensus.

Node Consensus

The *nodeConsen* can be obtained from literature or documentation for the consensus protocol. The purpose of *nodeConsen* is to find a threshold value for how many nodes need to functioning properly for the system to operate. This is often referred to as the byzantine node threshold or the number of nodes required for the system to function properly. If this does not exist, the measure can be obtained by running the ideal system and then removing nodes until the system no longer finds consensus or the consensus is incorrect. This measure can be a static number or a function based on the number of participating nodes.

Consensus Delay

The *consenDelay* can be recorded from the ideal system while the other measurements are be recorded. It is the time it takes the consensus system as a whole to find consensus. This time can be realized by the start time of when a transaction is requested and the time that consensus is found by the system. The flow for recording this is as followed:

1. A transaction is proposed to the consensus system
2. A time stamp is recorded at the time of the proposal
3. When the system finds consensus a time stamp is recorded
4. time stamps are reported and all time stamp variables are initialized

Data will be obtained for many transactions to get a large sample of consensus delay time stamps. These time stamps can be processed after recording to identify the difference between the two time stamps resulting in the time that each transaction took to find consensus. This data is then divided in two sets one set for calibration of the model and simulation during development and another set for the validation of the model and simulation.

Consensus State

Certain consensus protocols have periods of restructuring or organizing that occur when nodes go down or at various periods of time during operation. This special state of the system could be a leader selection process or anchor node assignment. This process is period where the system may delay transaction until the state is resolved based on each protocols own process. Once this state is resolved, the consensus protocol goes back to its normal transaction state. The measure *consenState* is the probability of at this moment in the simulation that the consensus system will go into this special state halting transactions until resolved. To obtain this value, literature of the consensus protocol can be utilized to identify if there is a periodic time when a special case occurs. This might happen when a change to the participating nodes occurs. If this is the case the *consensState* can be assigned to 1 when the network topology changes for the participating nodes.

State Length

For the special states, the time to process it needs to be identified. The variable *stateLength* will hold this value for each consensus protocol. This measure is the amount of time the consensus protocol takes to typically resolve the special state and continue processing transactions again. This measure will be obtained by identifying when the ideal system consensus protocol enters this special state and exits the special state. This measure will be sampled many times so that the data set can be fitted to a distribution to represent the possible values that represent the special state length.

Model and Simulation

The consensus system is modeled on a discrete event simulation platform. The nodes of the discrete event simulation either run as participating nodes of the consensus system or nodes that are required to complete the network topology. Each participating node runs the application that models the effects of the consensus protocol. The network simulator handles the flow of communication from each participating node to other parts of the system that have participating peers.

Discrete Event Simulation

To fully represent the consensus protocols' effects on different network topologies many simulation runs with samples of the input data will be needed. Each simulation run will give a glimpse of a possible result that could occur, doing this many times provides a better representation of the total system and how each of the modeled pieces has an affect on the total system results.

To complete these runs in a reasonable amount of time discrete event simulation is used. Using this paradigm of simulation allows the modelers to run the simulation experiments faster than real time. Each step of the simulation is now an event that occurs in the simulation rather than a time step. The events that would be represented in this project is a networked message. The discrete event simulation engine will schedule each message as the simulation runs based on how each modeled component affects the system. The scheduled events would also include the sampled time from the input distributions it would take to process the event. These event times are used to determine the order of subsequent events and also used for result reports that indicate the performance of the system.

Participating Peers

The participating peers are modeled as applications on nodes the modeled network simulation. The application includes the necessary inputs required to represent the consensus protocol that is being run. All of the input data for each state of the consensus protocol is stored as distribution libraries or variables in the application specified. As the simulation runs the participating peer will either begin a new transaction or participate in the consensus of an existing transaction. The participating peer will continue this operation until either a certain number of transactions have been processed or a period of time has elapsed.

Transactions are started when one of the peers proposes the next transaction to the system. It does this by initializing the transaction process then sends out messages to the participating nodes. The peer node schedules the message to every node to the simulation engine through the simulated network which will utilize existing routing libraries to handle the routing and the time for the message to reach the recipient. Based on the arrival time of the message to each node the simulation engine will order the events accordingly. Each event will be handled in the order they are scheduled in the event list. If possible the event list will be processed in parallel using high performance computing techniques.

As participating peer nodes receive messages of the consensus process. They will each internally record how many messages they have processed during the current consensus process. If they have processed enough messages based on the comparison of its current message count and *nMessage*. Then the nodes state will be assigned to complete so the total consensus system knows that this peer node has completed its consensus. If the peer node has not processed enough messages then it will schedule a response message with the simulation engine and associate the processed time as a sampled value from the *nodeDelay* distribution. The node will continue to process messages until the appropriate number of messages have been processed and its state has been changed to completed.

The peer that started the transaction will then monitor the system until *nodeConsen* number of participating nodes have a state of completed. Until that point the original participating node will contribute to the system in the same way as the other participating nodes. After enough participating nodes have reached a completed state, then either the same peer would start another transaction, another peer would start the transaction, or consensus system would change states. If the system changes states then transactions will be halted for period of time select by *stateLength.* The next event will be scheduled as a new transaction with a start time set after the delay selected from *stateLength.*

Calibration

Once the model is developed and functioning based on the observed data from the participating nodes, the total system will need to be observed on how close it is performing to the ideal system. The model and simulation is not expected to perform very close to the ideal system at first because there is potentially background traffic and other parts of the process that have not been fully accounted for. To better the model and simulation, calibration is used. Calibration is the process to tune the model so that it performs good enough compared to the ideal system. Tuning the model and simulation is done by increasing network background traffic to create more delay in the message processing. Calibration can also tune the distributions using constants as a scaling factor for selected input datasets as needed. This process is iterative and can use optimization methods and techniques to minimize the error between the model and simulation system and the observed data from the ideal system.

Validation

Once the model and simulation have been calibrated then the system is compared to the separate data set reserved for validation. Validation is the final check to ensure that the model does in fact represent the ideal system and not just the data of the ideal system that was used for calibration. This is a good check to identify if the model and simulation system was over calibrated to the point where it just mimics the observed data too close to the point that it does not represent the ideal system. This final validation check is also good in that it provides a final score of how close the model and simulation represent the ideal system. This test can be redone after another round of calibration if the dataset remains separate and the developers either reuse the data set for calibration or a different calibration dataset. If the validation step is repeated too many times it arguably could be considered part of the calibration step. Therefore it is best to minimize the validation attempts. The final scored value of the validation test is subjectively determined to be good or not. It is possible that models that score relatively bad in validation can end up still being very useful. The final result of the validation is an informative step that allows researchers to add confidence in the results of the simulated system.