ECE 428 MP2 Report

Name: Qihao Wang, Xianfan Gu
Netid: qw2, xianfan2

Repository URL:https://gitlab.engr.illinois.edu/xianfan2/ece428-sp20-mp2.git
Version number: 211d3c50ede3607f9ed40c1aedca311b0384d5ce

Codes with group are in the "graph" branch

- The cluster number you are working on

We use vm01 to run the introduction-service and vm09 to run the logger.
For the 20 scenario: we run the experiment on vm02 and vm03.
For the 100 scenario: we run the experiment on vm02 to vm08.

- Instructions for building and running your code

Copy the file from the git repository.
If the grader needs to run the logger, go to the “graph“ branch.

run the introduction-service on vm01 by typing python3 mp2_service.py 8888 <tx_rate>
use logger in vm09 by typing "go build logger.go" and "./logger 3000 > somefile.txt" if needed.

Start service:
“python3 mp2_service.py 3000 <tx_rate> [block_rate]”

To run the 20 nodes scenario (running on vm02-05):
in vm02-05, type :
"chmod 755 node.go"
"./test.sh"

To run the 100 nodes scenario (running on vm02-10):
in vm02,vm03...vm08, vm10, type:
"chmod 755 node.go"
"/test1.sh"

Log and plot for the results need to switch in the graph branch:
Start the logger and print output to the txt file:
"go run logger.go 3000 >> *.txt"
Follow the above steps to start the nodes

Each node will print the balance information, neighbor information and the longest chain length every 5 seconds.
To easily verify the correctness, we sum all the balances of the blockchain into the variable "checksum" and compare the "checksum" for every node.

How your nodes keep connectivity; how they discover nodes beyond the originally introduced ones, and how they detect failed nodes. You should justify why you think your design is robust to failures.

We create the basic blockchain based on the slides on lecture 15, the bitcoin white paper, "*Bitcoin: A Peer-to-Peer Electronic Cash System*" [1], and some tricks people discussed in campuswire. For the nodes connectivity, we chose gossip protocol network topology. When the node receive introduced nodes list from the service at the beginning of the service, it will establish the connection to these introduced nodes, and store them in a node memory pool, here we use a map to store the connections. The node would then send "REQUEST" to all introduced nodes for querying their list of introduced nodes so as to discover new neighbor nodes if the number of its neighbor less than 15.
The way to detect failure nodes is that we always read our tcp connection, it will return an error message("EOF") to indicate failure node and we can remove it from our neighbor list.
Since we will query our neighbors to give us their neighbors, when some of our neighbors crash and the number of our neighbors below the threshold, we can get a group of new nodes from the existing neighbor to keep the network robust.

How transactions are propagated. Describe the algorithm you are using, any parameters and how you arrived at them.

There are two ways to receive the transaction, from introduce-service or other nodes. When we receive the transaction from the introduce-service, we multicast it to our neighbor. when receiving transactions from other nodes, we will check whether we have already received this transaction by comparing the transaction ID and we will add this transaction to our mem pool only when it is a transaction we have never seen.

What information is logged and how you used this to generate the graphs. Please make sure that the logs you used in your experiments are checked into the git repository. If you wrote any scripts to analyze the logs please include them in the repo and describe how they work.

We run a logger on the VM-09 which will print the received message to the terminal and store terminal output to a txt file.

For part 1, we will log the bandwidth and the transaction delay. When we receive a message from the TCP channel, we will add the size of this message. We will count how many bytes this node per second and send it to our logger per second. When we receive a new transaction, we will send the timestamp in the message and the current time to the logger. As the experiment ends, we will use python to process the txt file. We sum the bandwidth per second from all nodes as our y-axis in graph and use time as our x-axis in graph. For the Delay graph, we use timestamp as the x-axis. In this case, for all transactions with the same timestamp, we can collect them together and sort the difference between the log time and timestamp to find the maximum delay, minimum delay, median delay and the number of transactions with the same timestamp to test the reachability.

For part 2, we will log the time between the service sending transactions and the blocks mining the transaction from the memory pool as the time each transaction takes to appear in a block. We send the timestamps of the nodes and the transactions to the logger once the miner put the transaction into the block. The x-axis can be the timestamps of the transactions and the y-axis can be the delay of the transactions (in sec units). Furthermore, we send the block hash and the timestamps of the nodes to the logger once the miner receives the solved message, then we log the block hash and the timestamps at the handler where the node can receive the broadcast blocks. The block propagate throughout the network should be the maximum time interval from the miner receiving the solved message to the node receiving the block. For this plot, x-axis is the order of the propagate blocks and the y-axis is the time delay of the transactions (in sec units). For the chain splits, we log the height and the parent of the mined blocks when the block is inserted into the blockchain. To plot the longest split of each chain split, we count all the different blocks with the same height, then backtracking the parents of two fork chains in the same level until two parents are matched. The x-axis in the plot is the number of mined blocks, the y-axis is the longest split at the height of the mined blocks.

What the process is for validating transaction validity and collecting them into blocks.

To check the validity of transactions, we check whether the source account has enough balance, if it has enough balance, we will put it into the block.

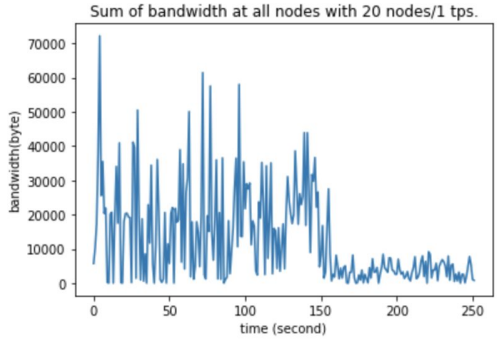
How new blocks are propagated and imported. Discuss particularly what happens during a chain fork

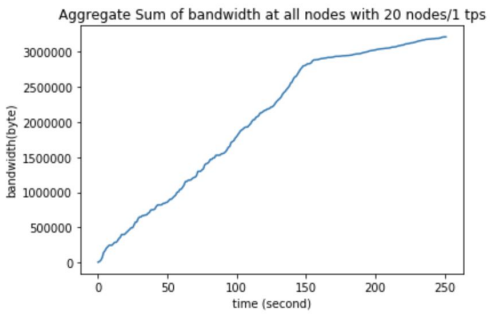
When the miner receives a solved message, it will check whether the block is in the queue of the received block and the pending blocks (some blocks are still waiting for the reply from the service). Then the miner will multicast the new block to its neighbor nodes if the block satisfies that it does not appear in the queue of received blocks and the pending blocks. When the block is propagated to a node, it needs to know whether the block is new and decide to multicast the block to its neighbors, and send the verify message to the service for verifying the block if the block is new for the node . The block can be inserted into the blockchain if the block is verified OK from the service, and the hash of the block is within the threshold of the target. To insert the block into the blockchain, the node should check the parent block of the block is in the blockchain, otherwise it would be stored in an orphan buffer in the blockchain, and waiting until its parent block joins into the block chain. To prevent the cause of the chain fork modifying the state of the blockchain, we only update the parent block of the tip of the longest tree. However, if the length of the chain fork has two blocks longer than the longest chain, then the state will be modified by the chain fork. Thus, whether the chain fork can change the state of the blockchain depends on the propagation delay of the blocks.

Graph:

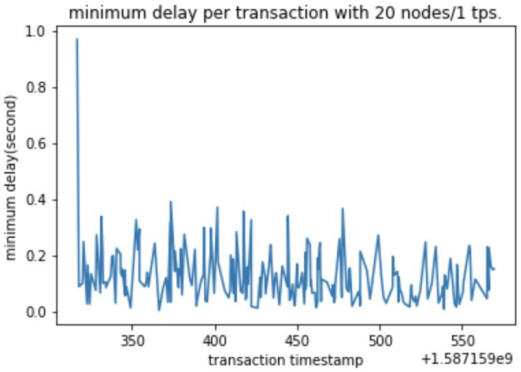
20 nodes experiment:

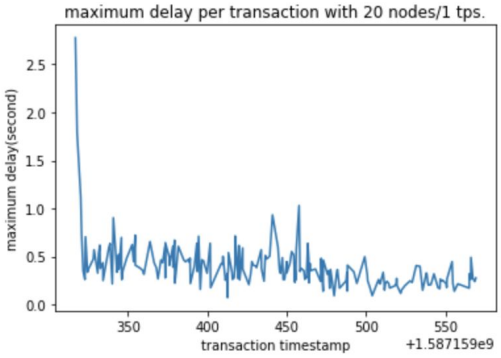
Bandwidth:

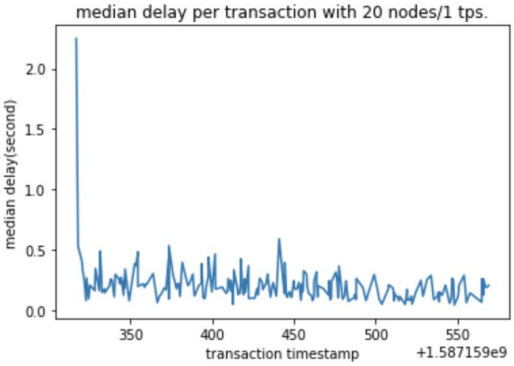


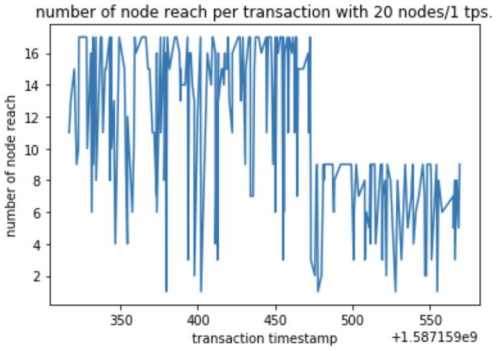


Transaction Delay





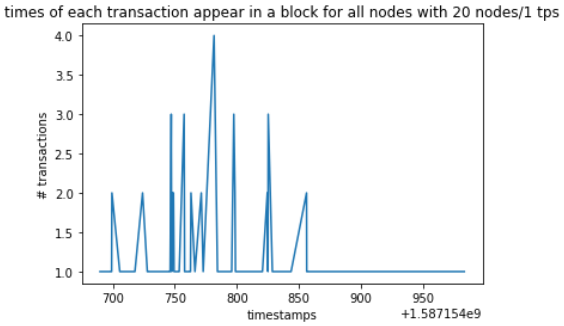


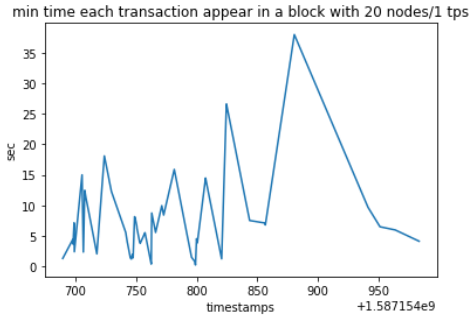


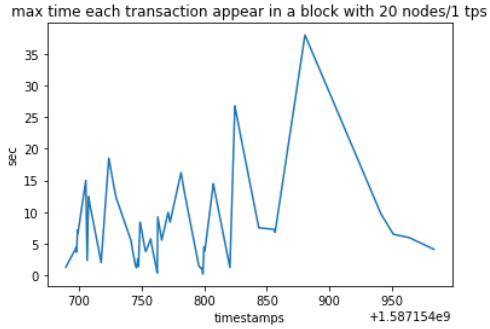
Part II

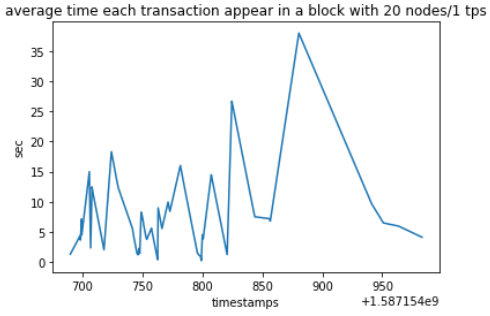
20 nodes experiment (block rate 0.1):

Time interval of the transaction to appear in a block

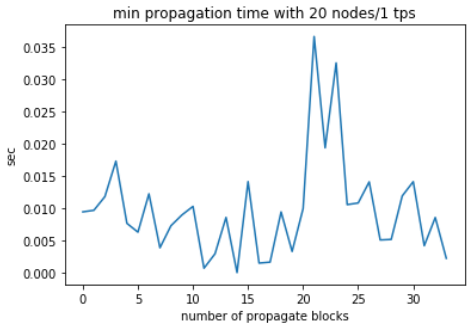


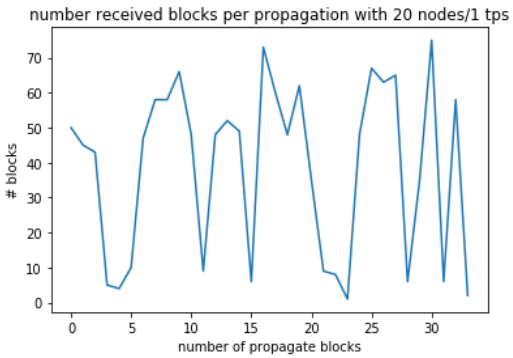


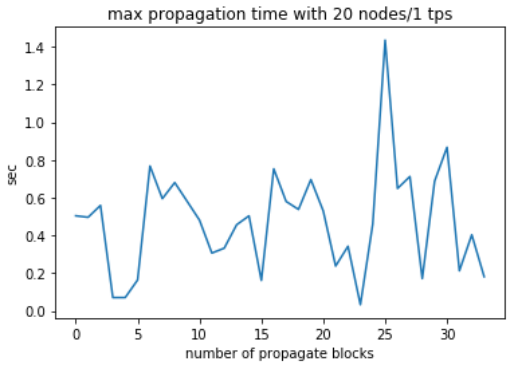


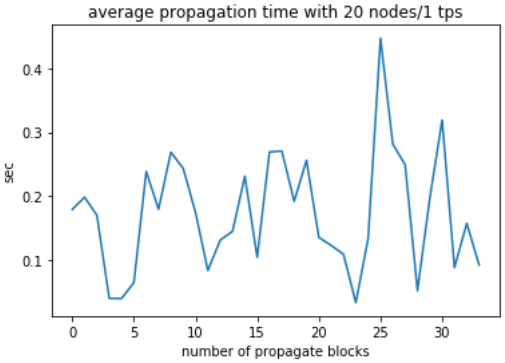


20 nodes experiment (block rate 0.1):
Propagation of the solved block

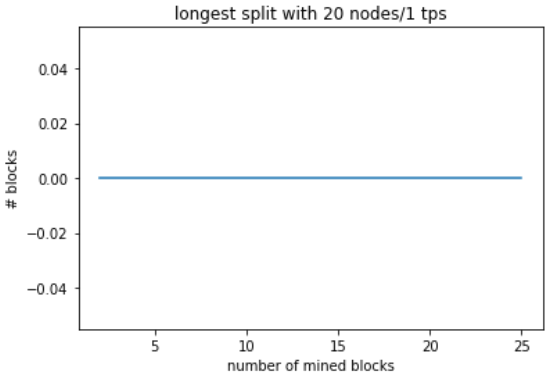






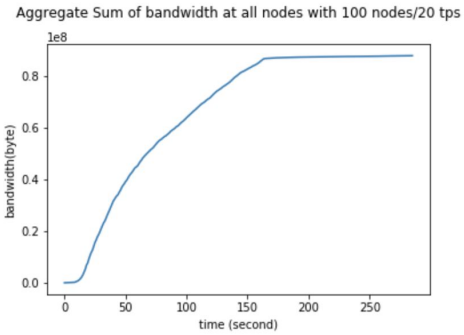


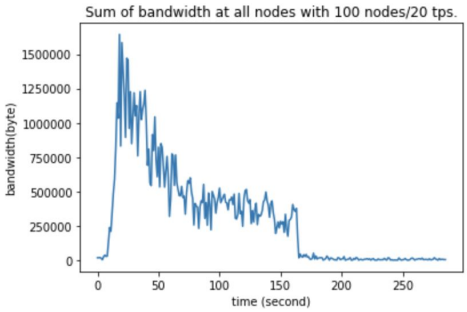
Longest split of chain forks



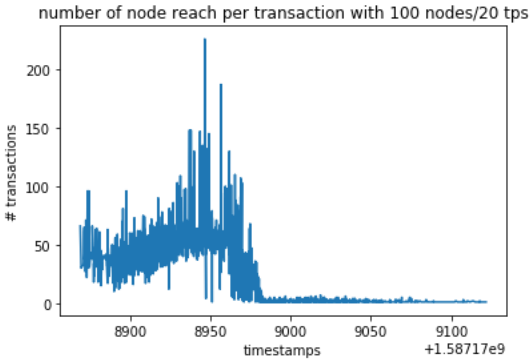
100 nodes experiment:

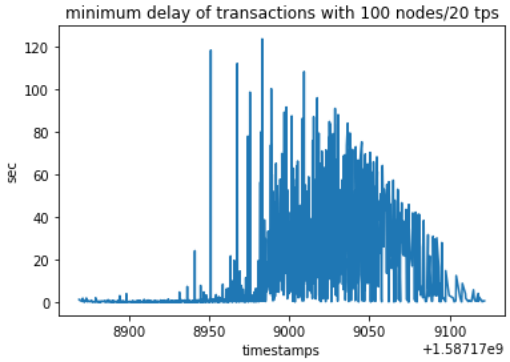
Bandwidth:

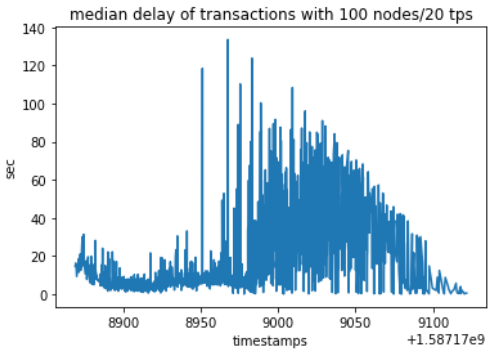


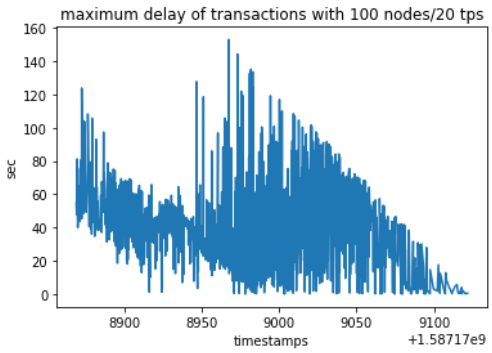


Delay:



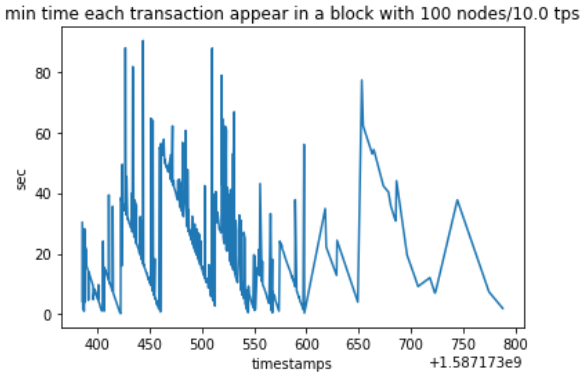


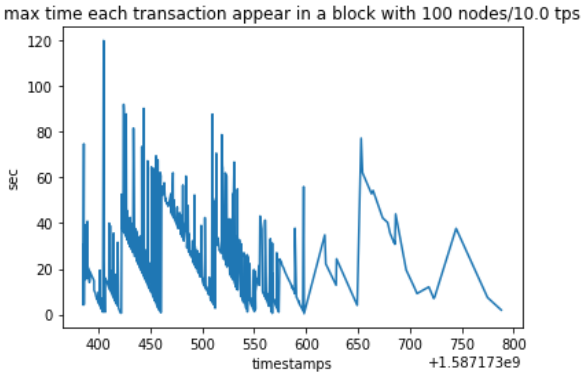


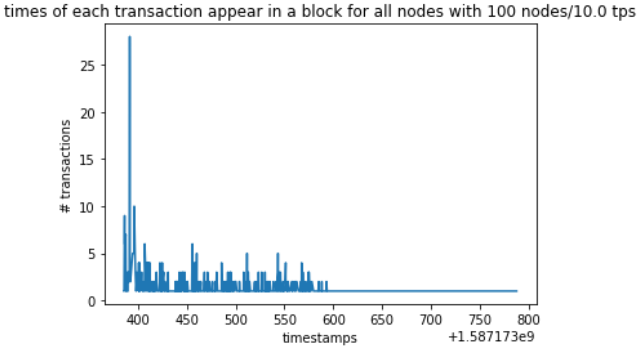


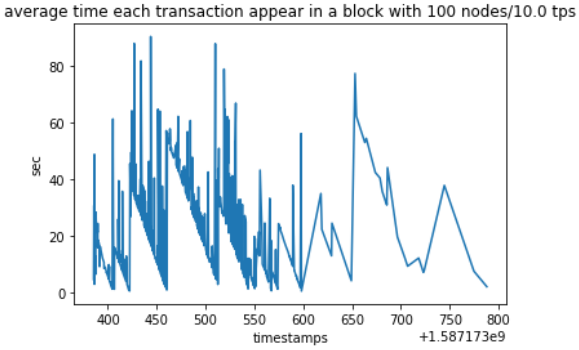
Part II

100 nodes experiment (tx rate 10.0, block rate 0.05):
Time interval of the transaction to appear in a block

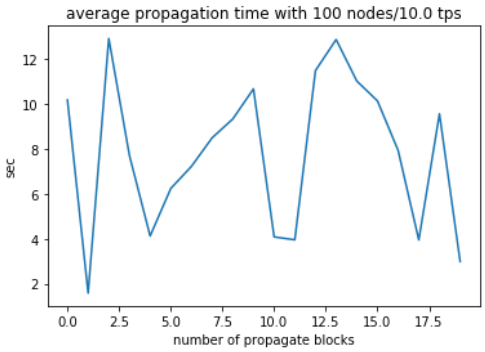


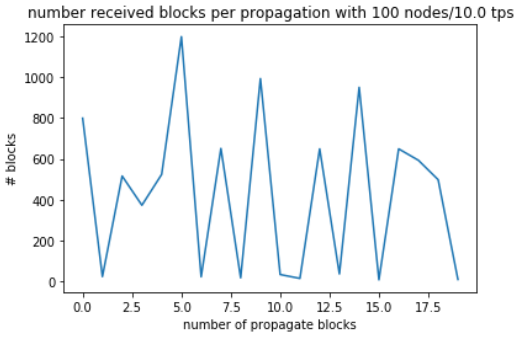


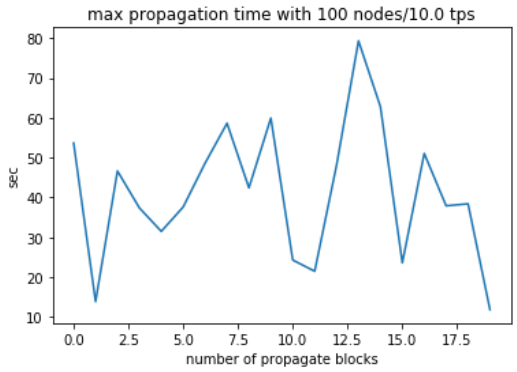


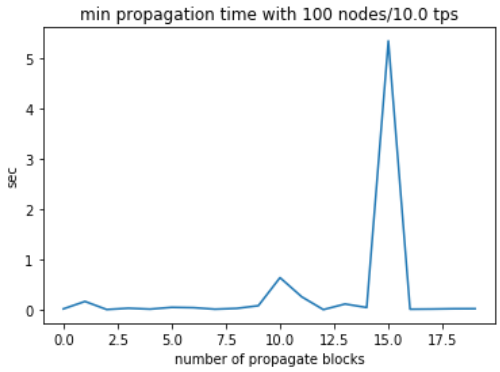


100 nodes experiment (tx rate 10.0, block rate 0.05):
Propagation of the solved block

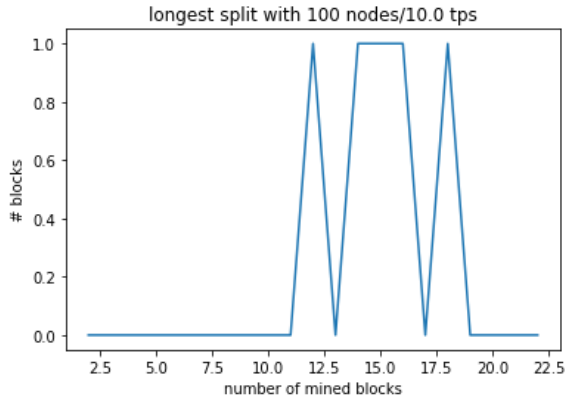








Longest split of chain forks



Analysis:

For the 20 nodes experiment, our printed state balance and longest chain matches our expectation. From the graph, it seems some transactions are not received by all nodes and we think it is because the workload for the threads is a little bit heavy. The propagation delay graph and bandwidth seem reasonable and in an acceptable range. Both the delay time of transaction broadcast and the block propagation are decreased as the nodes of the network system and the service transaction rate increase. When half of the nodes are terminated, the block propagation delay of the rest of nodes would increase, and the transaction delay would also increase.

For the 100 nodes experiment, it does not match our expectation. We think it is because the network traffic is so congested that a lot of packages drop. Also, from the bandwidth

graph we find that the bandwidth decreases sharply when typing “thanos”. We think it is because we only allow each neighbor to have at most 15 neighbors and if we close 50 nodes suddenly, some nodes may lose all incoming connections and outcoming connections. As we look at the log file, we find lots of strings lose a few characters so that we cannot decode it well and that is why our graph in 100-nodes-experiment looks weird.

Reference

[1]Satoshi Nakamoto, ”Bitcoin: A Peer-to-Peer Electronic Cash System”, *www.bitcoin.org,* https://bitcoin.org/bitcoin.pdf