ECE 428/CS 425 Distributed Systems
Spring 2020
MP 2
Cluster: g22
URL: https://gitlab.engr.illinois.edu/asahdev2/ece428
Revision Number: 5b1f9800b773cffd98d9819df6d19f94dfb16463

Navid Mokhlesi, navidm2
Amber Sahdev, asahdev2

For MP2 we extensively used Go's concurrency features such as Go routines and channels.

To build the code run the following within the MP2 directory of the repository

```
> make
```

After running the makefile, we can run the node as follows:

```
> ./main [node name] [node port number]
```

To run a bunch of services at once, we have designed a launcher service in go which can be launched as such:

```
> go build launcher.go; ./launcher [Num Nodes] [Time to Live] [Base Port to Start Launching
Nodes From]
```

We fetch to service address and port from MP2/serviceAddr.txt

We also need to install certain python dependencies to generate graphs as follows:

```
> pip3 install matplotlib
> pip3 install numpy
```

The python script to generate graphs can be run as follows

```
> python3 graph.py
```

We generated three kinds of graphs: Bandwidth, Reachability, and Propagation. The Bandwidth graph represents the number of bytes we decode from all our TCP connections combined. The Reachability graph represents the number of nodes/peers a transaction has reached. The Propagation graph represents how the minimum, maximum, and median time a given transaction takes to propagate to all the nodes/peers it propagated to. Every peer writes to a log file about transaction metadata, such as timestamp of transaction, time the peer received it, transaction ID, and the bandwidth used in exchange of blocks, transactions, neighbor metadata.

The minimum propagation delay in the propagation delay graph corresponds to when the first peer receives the transaction that did not receive that transaction from the mp2_service.

System Design:

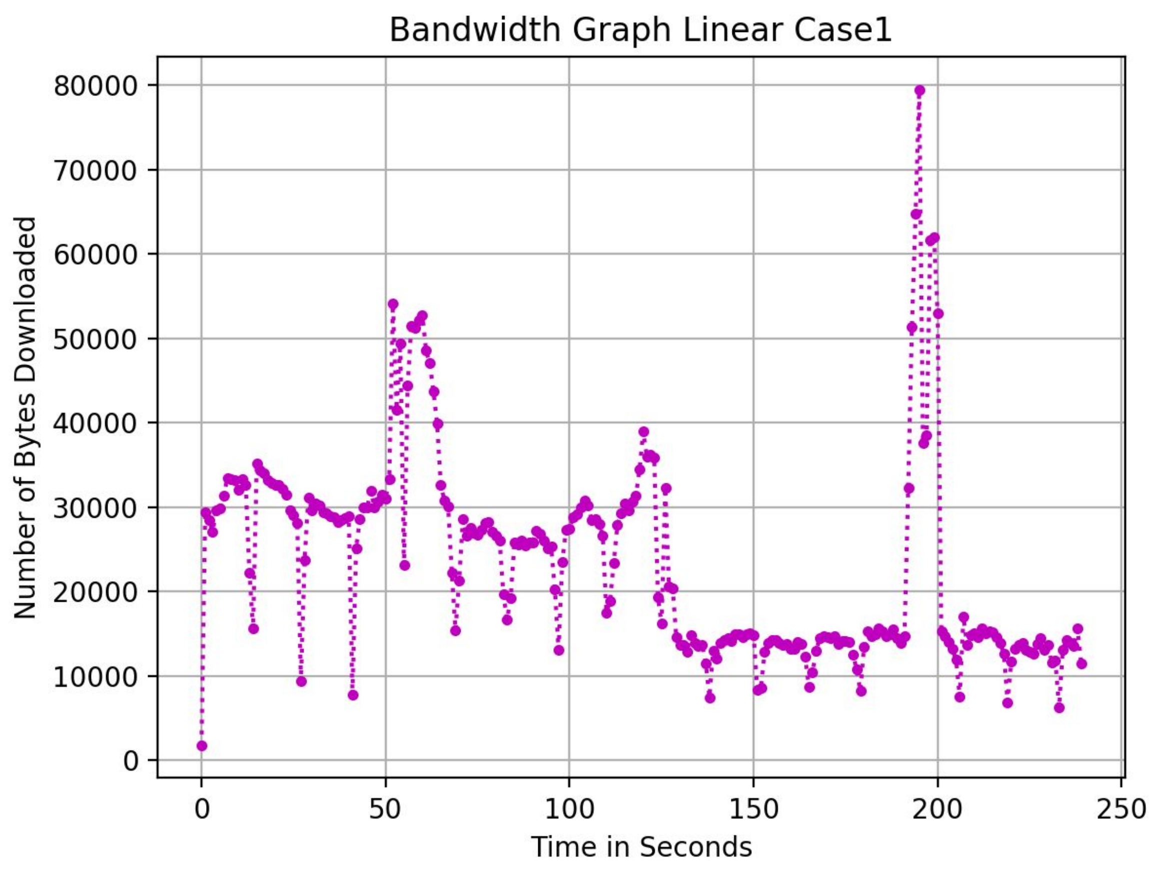
Our application is designed to support a large (100+) node blockchain system that uses pull gossip to communicate information between nodes. Our Pull gossip handshake is done such that each node polls it's neighbors in a round robin fashion (with 1 second polling period between nodes). When a node receives a poll message, it replies with what transcationIDs, node names, and blockIDs it has received since the last time it was polled by the initiating node. The initiating node then requests the items that it has not already received from other nodes. The polled service then replies with all the requested data values. Additionally there is a desired number of neighbors computed by each node, and a node attempts to keep connection to at least Ceiling(log_2(number of nodes in network))) nodes at any given time using the gossiped list of all nodes in the network. This is robust to failures because when neighbors are killed new ones are established using the gossiped node list.

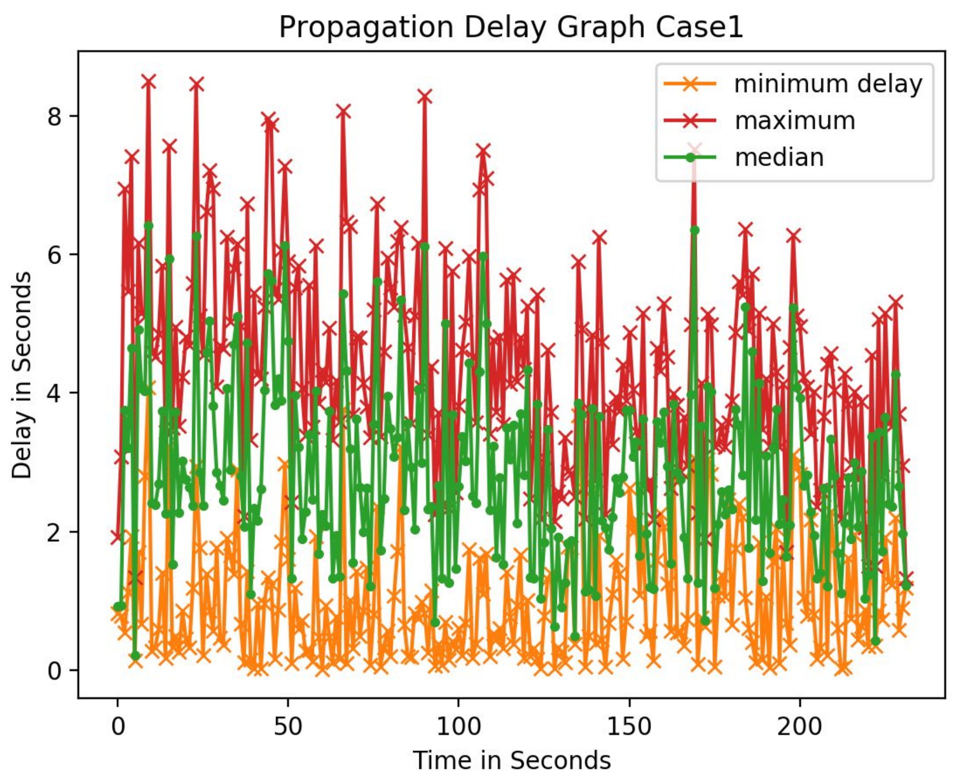
Our blockchain manages it's blockchain ledger using pull gossip to communicate mined blocks and pending transactions across the network. Nakamoto consensus on longest chain is used, and leverages the mp2_service.py in place of computing our own proof of work. Each node in the system maintains a set of transactions included in the current longest blockchain, and the current longest blockchain is only switched, if an incoming verified block exceeds the current longest max chain height. In this event, a rollback of transactions from the commited list occurs by traveling up the previous longest chain until meeting the common ancestor of the two chains. The new transactions are applied as the algorithm travels down the new longest chain, and the new pending transaction list is updated to reflect the change of chains.

Lifecycle of a block: When a node's pending transaction list reaches a target number of elements, it begins mining that block (assuming the transactions are valid as a whole) by sending solve to MP2 service. If a node successfully mines a block, it will add it to it's list of blocks and then begins the process of propagating that block across the network. When a node pulls a block from another node, it puts it into a set of blocks to be verified. A received block is added to the list of blocks, and that block also begins propagating across the network. Whenever a block is added to the list of blocks, it is verified, if the block is the new leaf of the longest chain, then the node removes the transactions present in that block from the pending transactions list, and continues gossiping the new transaction list. The block the current node is trying to mine is always the descendent of the longest chain leaf block, with ties for longest chain leaf going to the block which was verified first. Block propagation throughout our network is handled by the gossip protocol, as such, the bandwidth, delay, and reachability graphs were sufficient to fully describe how long a block takes to be propagated throughout the network after it is mined. In part 2 our transaction to block rate was limited not by bandwidth but by the block generation code of the mp2_service, therefore we saw all valid seen transactions appear in the block on average every 1-2 minutes.

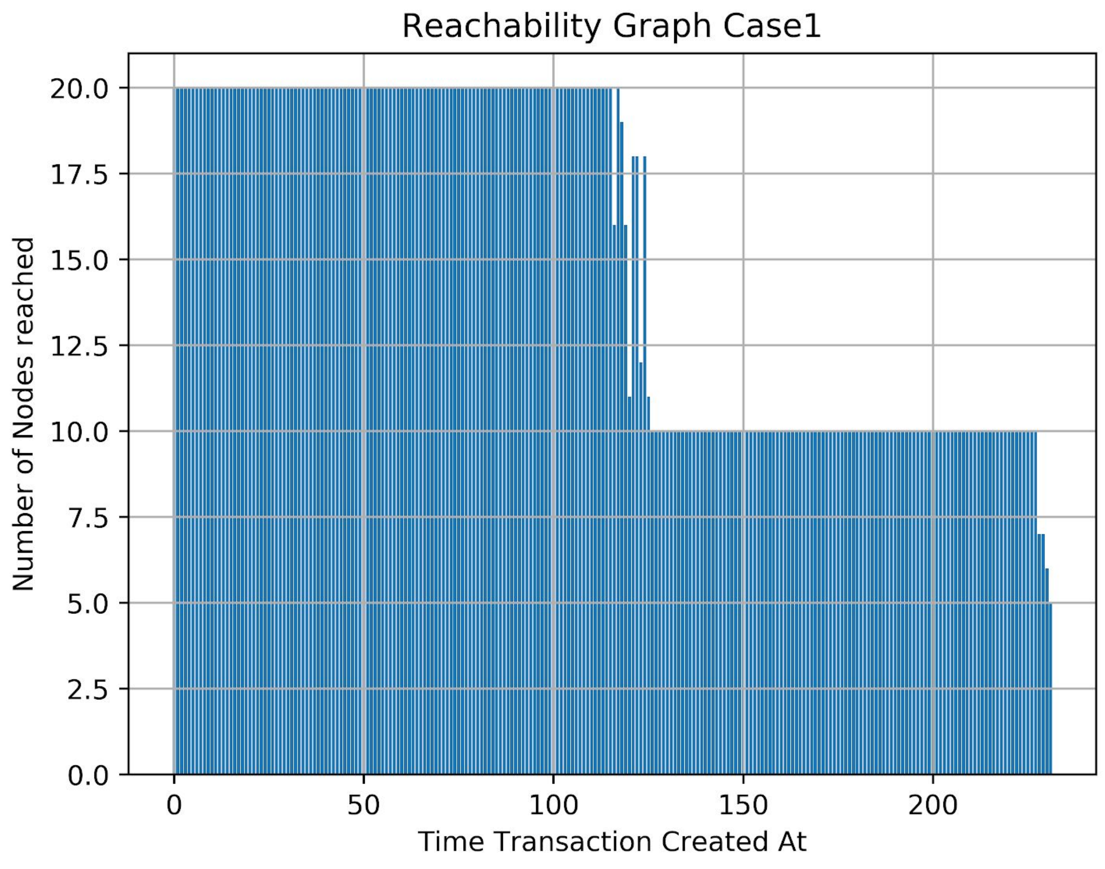
Graphing and Evaluation:
The data used to generate these plots is in *MP2/case1/* and *MP2/case2/*

Case 1: 20 nodes, 1 transaction generated per second



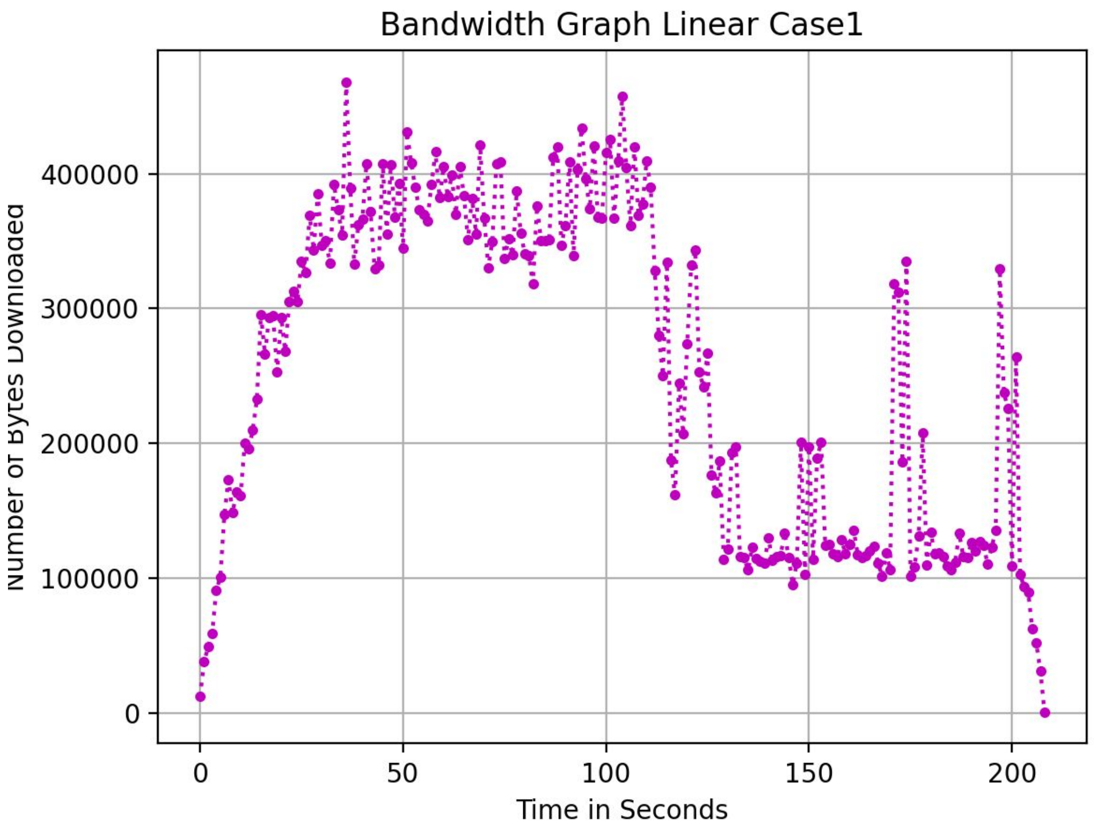


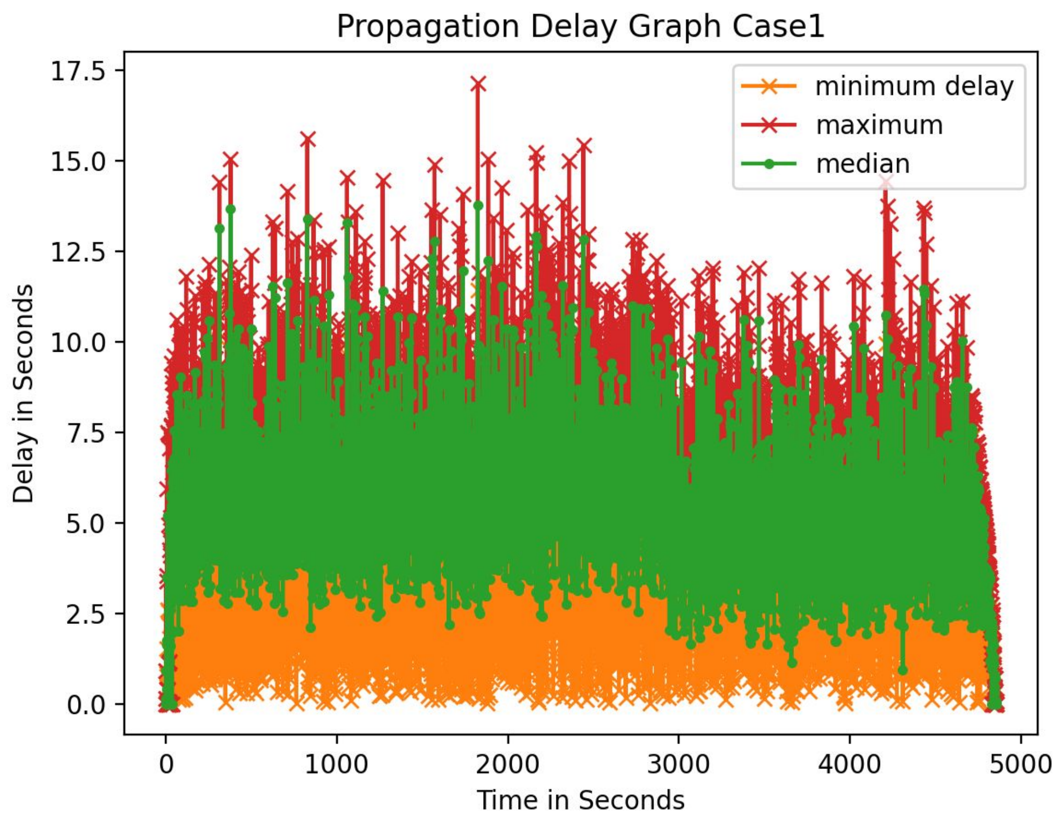
Propagation Delay Graph Case1
minimum delay
maximum
median
Delay in Seconds
Time in Seconds
0
2
4
6
8
0
50
100
150
200

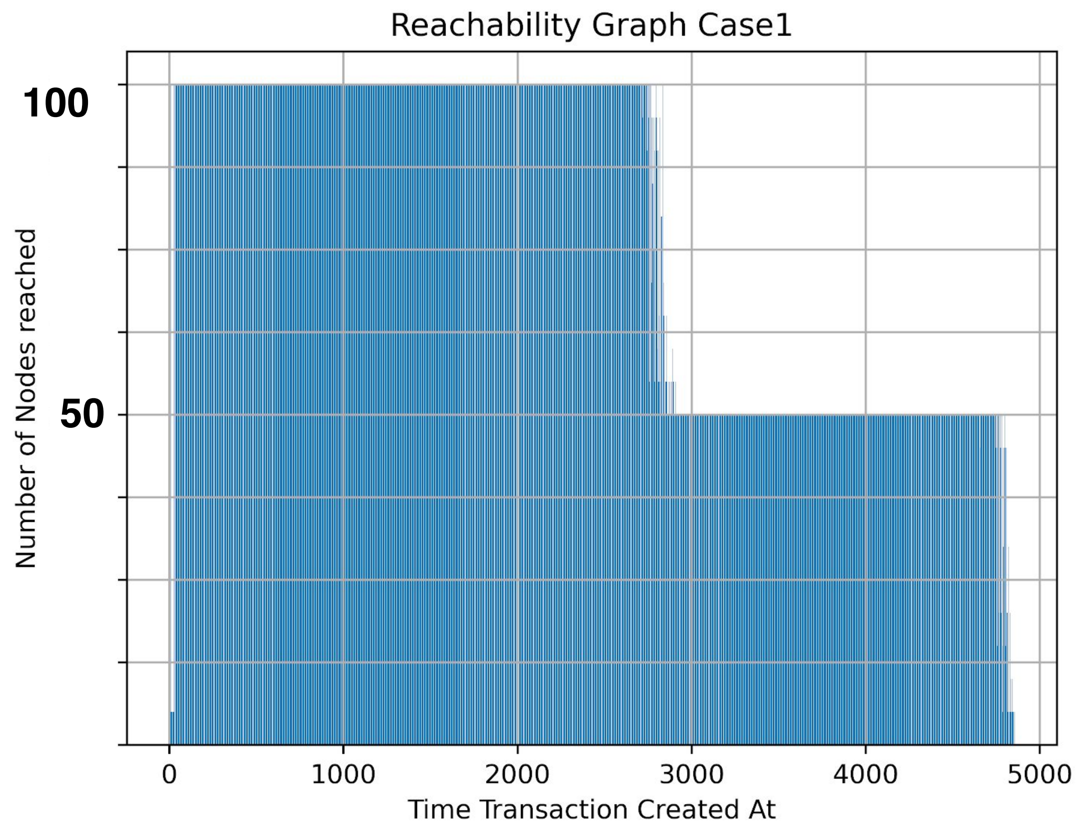


Reachability Graph Case1
Number of Nodes reached
Time Transaction Created At
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
50
100
150
200

Case 2: 100 nodes, 20 transactions per second




Propagation Delay Graph Case1
minimum delay
maximum
median
Delay in Seconds
Time in Seconds
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
1000
2000
3000
4000
5000


Reachability Graph Case1
100
50
Number of Nodes reached
Time Transaction Created At
0
1000
2000
3000
4000
5000