MP1 REPORT

- Zonglin Peng (zonglin7)
- Huiming Sun (huiming5)

The cluster number we are working on is `g03`

GitHub Link

https://github.com/cs425-ece428/mp1-monad-mp1

INSTRUCTIONS FOR BUILDING AND RUNNING

```
# Quick Build
bash ./script/unix/mp1/quick_build.bash
# Release Build
bash ./script/unix/mp1/build.bash
# Usage

# 1 nodes for local test
./bin/mp1 A 8080 ./lib/mp1/config/1/config_a.txt

# 2 nodes for local test
./bin/mp1 A 8080 ./lib/mp1/config/2/config_a.txt
./bin/mp1 B 8081 ./lib/mp1/config/2/config_b.txt

# 3 nodes
./bin/mp1 A 8080 ./lib/mp1/config/config_a.txt
./bin/mp1 B 8081 ./lib/mp1/config/config_b.txt
./bin/mp1 C 8082 ./lib/mp1/config/config_c.txt
```

```
# just stdout
python3 -u ./script/unix/mp1/gentx.py 0.5 | ./bin/mp1 A 8080
./lib/mp1/config/3/config_a.txt 2> /dev/null
python3 -u ./script/unix/mp1/gentx.py 0.5 | ./bin/mp1 B 8081
./lib/mp1/config/3/config_b.txt 2> /dev/null
python3 -u ./script/unix/mp1/gentx.py 0.5 | ./bin/mp1 C 8082
./lib/mp1/config/3/config_c.txt 2> /dev/null

# stderr to file
python3 -u ./script/unix/mp1/gentx.py 0.5 | ./bin/mp1 A 8080
./lib/mp1/config/3/config_a.txt 2> /tmp/a.log
python3 -u ./script/unix/mp1/gentx.py 0.5 | ./bin/mp1 B 8081
./lib/mp1/config/3/config_b.txt 2> /tmp/b.log
python3 -u ./script/unix/mp1/gentx.py 0.5 | ./bin/mp1 C 8082
./lib/mp1/config/3/config_c.txt 2> /tmp/c.log

# json log
python3 -u ./script/unix/mp1/gentx.py 0.5 | LOG=json ./bin/mp1 A
8080 ./lib/mp1/config/3/config_a.txt 2> /tmp/a.log
python3 -u ./script/unix/mp1/gentx.py 0.5 | LOG=json ./bin/mp1 B
8081 ./lib/mp1/config/3/config_b.txt 2> /tmp/b.log
python3 -u ./script/unix/mp1/gentx.py 0.5 | LOG=json ./bin/mp1 C
8082 ./lib/mp1/config/3/config_c.txt 2> /tmp/c.log

# trace log
python3 -u ./script/unix/mp1/gentx.py 0.5 | LOG=trace ./bin/mp1 A
8080 ./lib/mp1/config/3/config_a.txt 2> /tmp/a.log
python3 -u ./script/unix/mp1/gentx.py 0.5 | LOG=trace ./bin/mp1 B
8081 ./lib/mp1/config/3/config_b.txt 2> /tmp/b.log
python3 -u ./script/unix/mp1/gentx.py 0.5 | LOG=trace ./bin/mp1 C
8082 ./lib/mp1/config/3/config_c.txt 2> /tmp/c.log

# 8 nodes
python3 -u ./script/unix/mp1/gentx.py 5 | LOG=json ./bin/mp1 node1
8080 ./lib/mp1/config/8/config_1.txt 2> /tmp/1.log
python3 -u ./script/unix/mp1/gentx.py 5 | LOG=json ./bin/mp1 node2
8081 ./lib/mp1/config/8/config_2.txt 2> /tmp/2.log
python3 -u ./script/unix/mp1/gentx.py 5 | LOG=json ./bin/mp1 node3
8082 ./lib/mp1/config/8/config_3.txt 2> /tmp/3.log
```

```
python3 -u ./script/unix/mp1/gentx.py 5 | LOG=json ./bin/mp1 node4
8083 ./lib/mp1/config/8/config_4.txt 2> /tmp/4.log
python3 -u ./script/unix/mp1/gentx.py 5 | LOG=json ./bin/mp1 node5
8084 ./lib/mp1/config/8/config_5.txt 2> /tmp/5.log
python3 -u ./script/unix/mp1/gentx.py 5 | LOG=json ./bin/mp1 node6
8085 ./lib/mp1/config/8/config_6.txt 2> /tmp/6.log
python3 -u ./script/unix/mp1/gentx.py 5 | LOG=json ./bin/mp1 node7
8086 ./lib/mp1/config/8/config_7.txt 2> /tmp/7.log
python3 -u ./script/unix/mp1/gentx.py 5 | LOG=json ./bin/mp1 node8
8087 ./lib/mp1/config/8/config_8.txt 2> /tmp/8.log
```

Verbose Mode

```
LOG=trace
```

JSON Mode

```
LOG=json
```

Command Line Arguments

Each node must take three arguments. The first argument is an identifier that is unique for each node. The second argument is the port number it listens on. The third argument is a configuration file – the first line of the configuration file is the number of other nodes in the system that it must connect to, and each subsequent line contains the identifier, hostname, and the port no. of these nodes. Note the configuration file provided to each node will be different (as it will exclude the identifier, hostname and port of that node). For example, consider a system of three nodes with identifiers node1, node2 and node3, where a node runs on each of the first 3 VMs in your group (say g01), and each node uses port no. 1234. The configuration file provided to node1 should look like this:

```
2
node2 fa21-cs425-g01-02.cs.illinois.edu 1234
node3 fa21-cs425-g01-03.cs.illinois.edu 1234
```

The configuration file for the second node will look like this:

```
2
node1 fa21-cs425-g01-01.cs.illinois.edu 1234
node3 fa21-cs425-g01-03.cs.illinois.edu 1234
```

And so on. We will use our own configuration files when testing the code, so make sure your configuration file complies with this format.

Each node must listen for TCP connections from other nodes, as well as initiate a TCP connection to each of the other nodes. Note that a connection initiation attempt will fail, unless the other node's listening socket is ready. Your node's implementation may continuously try to initiate connections until successful. You may assume no node failure occurs during this start-up phase. Further ensure that your implementation appropriately waits for a connection to be successfully established before trying to send on it.

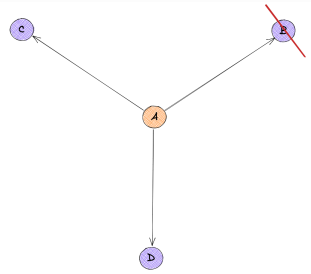
Note: make sure your node can run using the EXACT command given below.

```
./mp1_node {node id} {port} {config file}
```

DESIGN DOCUMENT

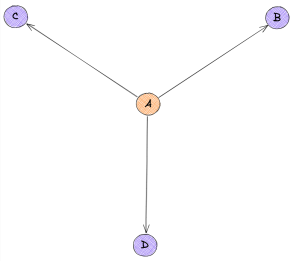
We use a combination of ISIS algorithm and R-Multicast to ensure Reliable Total-Ording. The ISIS algorithm can guarantee Total-Ording, R-Multicast to ensure reliable Multicast. In addition, we also assume that once a node loses its TCP connection, it will be evicted from the Group (A failed node will not become alive again).

Proof of correctness



Recipent Crash

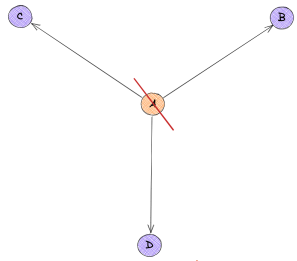
A will eject crash Node from Group



Ask Seq Num

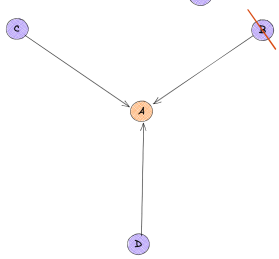
```
type TOAskProposalSeqMsg struct {
        SrcID string `json:"src"`
        MsgID string `json:"msg_id"`
        Body  []byte `json:"body"`
}
```

Use R-Multicast to send a request to ask
what should be the Seq Num for a given Msg



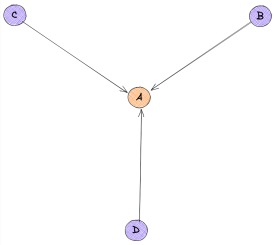
Sender Crash

R-Multicast could make sure "all or nothing"



Recipent Crash

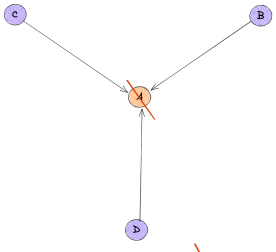
Unicast could make sure "one or nothing"



Wait Seq Num

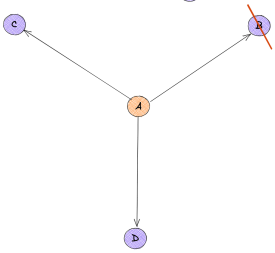
```
type TOReplyProposalSeqMsg struct {
        ProcessID   string `json:"pid"`
        MsgID       string `json:"msg_id"`
        ProposalSeq uint64 `json:"proposal_seq"`
}
```

Use Unicast to reply the ask Msg



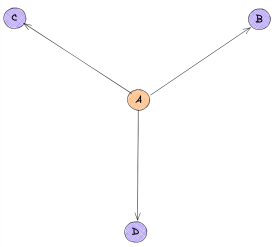
Sender Crash

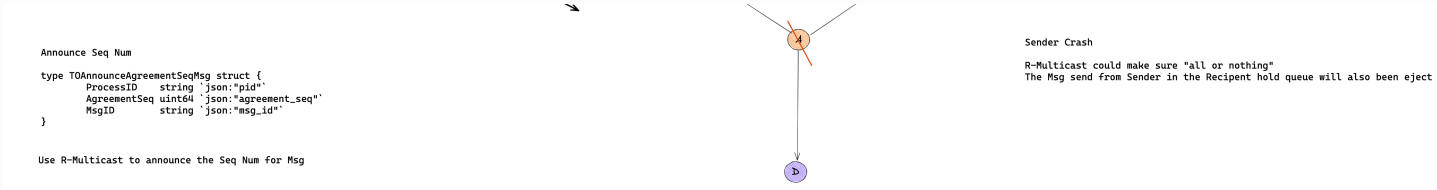
All Recipent will eject Sender from Group
The Msg send from Sender in the Recipent hold queue will also been eject



Recipent Crash

R-Multicast could make sure "all or nothing"
Sender will eject Crashed Recipent from Group





ISIS

We follow the instructions in the following pseudo code to implement ISIS.

ISIS algorithm for total ordering of messages

Let $g=\{p_1,p_2,p_3,...p_n\}$ be a closed group of processes that multicast all messages to the entire group. The following algorithm, executed at each process p_i in g will ensure that all processes in g will deliver all messages in the same order.

On initialization: $s_i = 0$; $counter_i = 0$;

On multicast of message m to all processes in g
  $counter_i := counter_i + 1$;
  B-multicast(<m,$counter_i$,i>);

On B-deliver of message <m,m_{id},j> from p_j (where $1 \le j \le n$)
  $s_i := s_i + 1$;
  Send <m_{id},s_i> to p_j;
  Put <m,m_{id}, j, s_i, i, undeliverable> in hold-back queue;

On receive(<m_{id},s_j>) from p_j (where $1 \le j \le n$)
  Add <s_j,j> to list of suggested sequence numbers for message m_{id};
  if we have received sequence number from all processes in g then
    <s_k,k> := highest sequence number in list (suggested by p_k);
    // choose smallest possible value for k if there are multiple suggesting this sequence #
    B-multicast(<m_{id}, i, s_k, k>);
  end-if

On B-deliver of message <m_{id}, i, s_k, k> (where $1 \le k \le n$)
  $s_i := \max(s_i,s_k)$
  Modify message with id <m_{id}, i> on hold-back queue as follows:
    change proposed sequence number to s_k;
    change process that suggested sequence number to k;
    change undeliverable to deliverable;

On addition to queue or changing of element in queue
  // note that all elements have the following format: <m,m_{id},j,s,k,status>, where
  // m is the message, m_{id} is the message id of m, j is the process that sent m,
  // s is the suggested sequence number, k is the process that suggested s (or k is the
  // suggesting process), and status is either deliverable or undeliverable.
  Sort such that message with smallest sequence number is at the head
  If two sequence numbers are the same then
     place any undeliverable messages at the head
     to break further ties, place message with smallest suggesting process # at the head
  end if
  While message at head of queue has status deliverable do
    deliver the message at the head of the queue
    remove this message from the queue
  end while

Terminology / System Architecture

Node

`lib/mp1/multicast`

Each Node is an independent individual that exists in the Group;
This Node and other Nodes perform full-duplex TCP communication according to the configuration in the configuration file.
Each Node has a unique Node ID in the Group to indicate the identity of the process.

Group

`lib/mp1/multicast`

Group is a collection of Node.
Group encapsulates `Unicast`, `B-Multicast`, `R-Multicast` and `TO-Multicast`.

Config

`lib/mp1/config`

Parse the configuration file format

Dispatcher

`lib/mp1/dispatcher`

A simple fully match router implementation.

Metrics

- Serialization bandwidth and delay struct
- Log bandwidth and latency

Transaction

- The logical of the transaction
- Parse transactions raw string
- Register transaction processing function

Retry

`lib/retry`

Retry call f every interval until the maximum number of attempts is reached.
If the incoming attempts is 0, retry forever

Broker

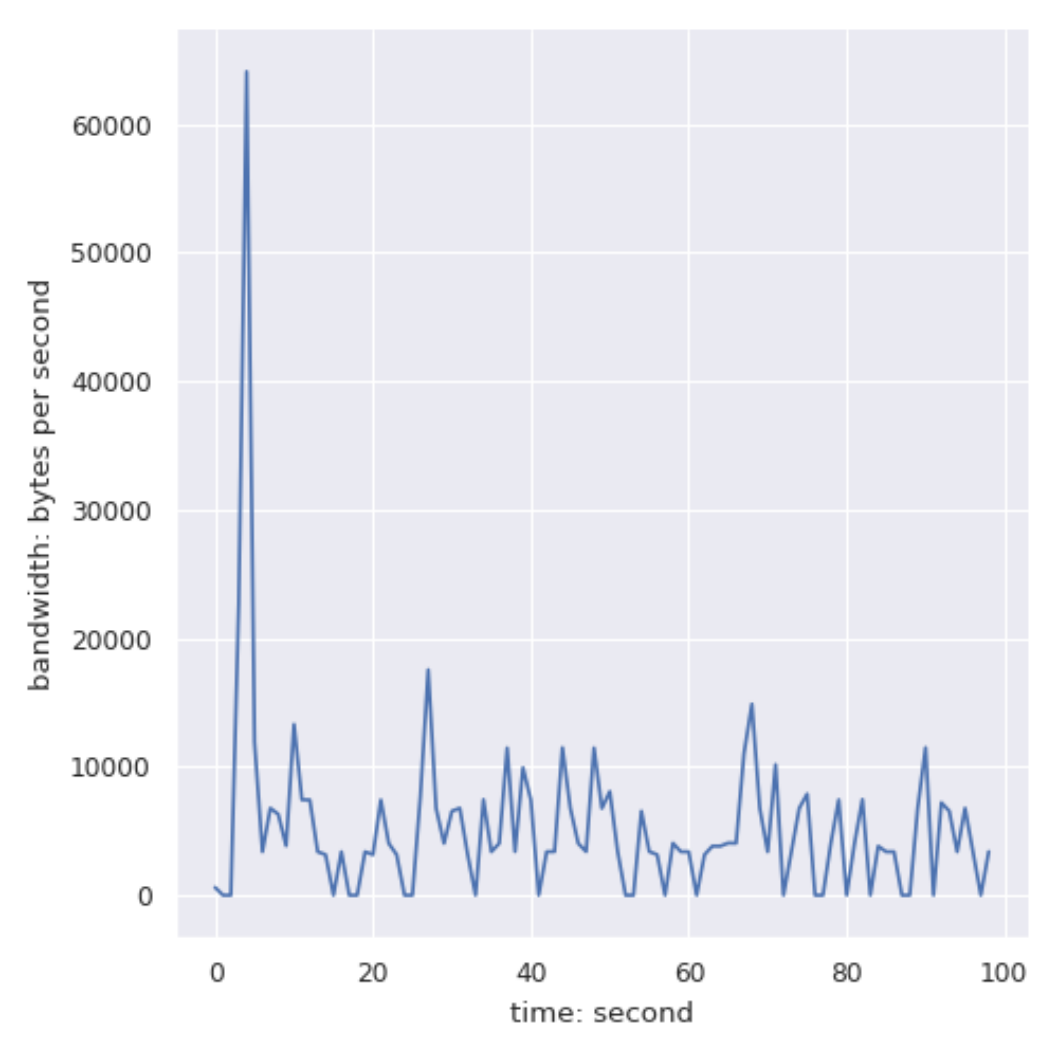
`lib/broker`

A one to many proxy for channel

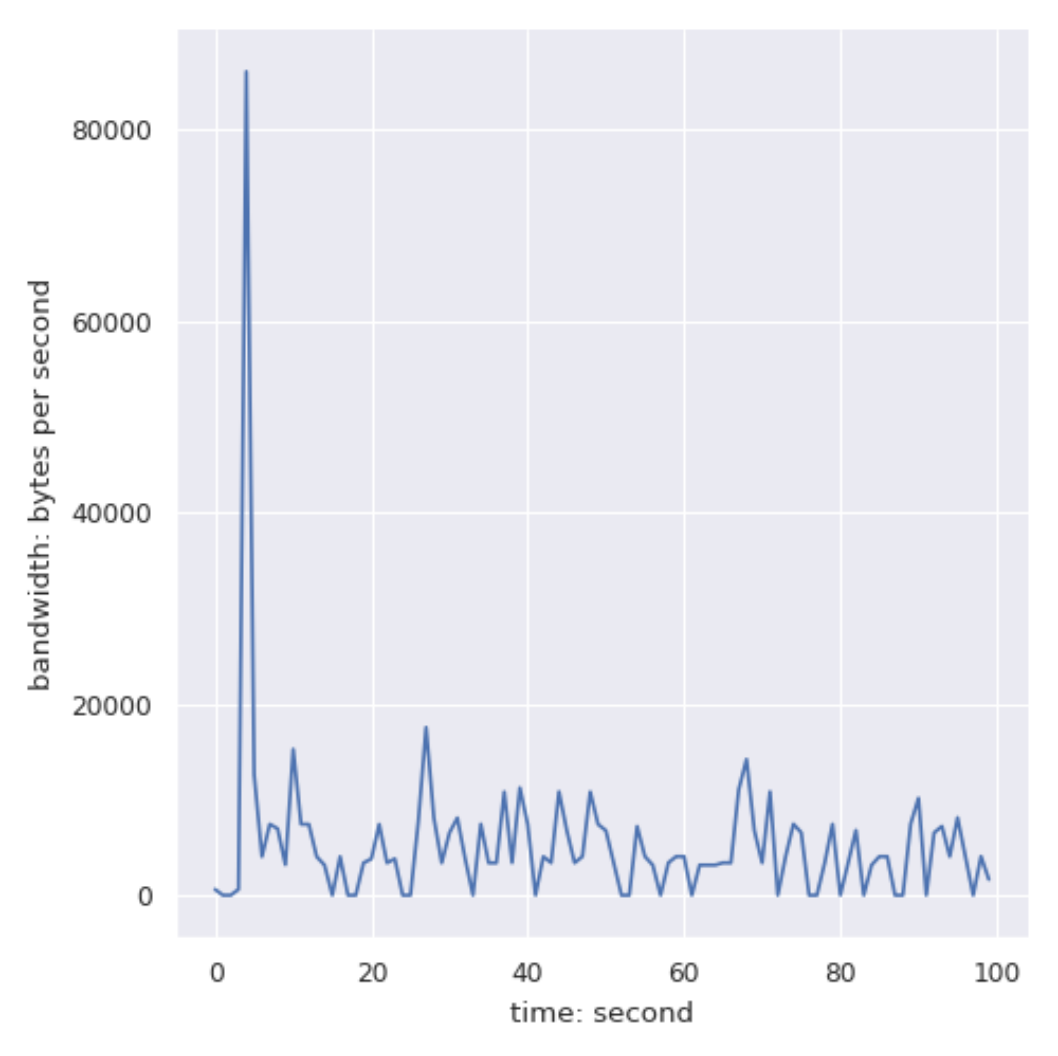
GRAPHS OF THE EVALUATION

3 nodes, 0.5 Hz each, running for 100 seconds

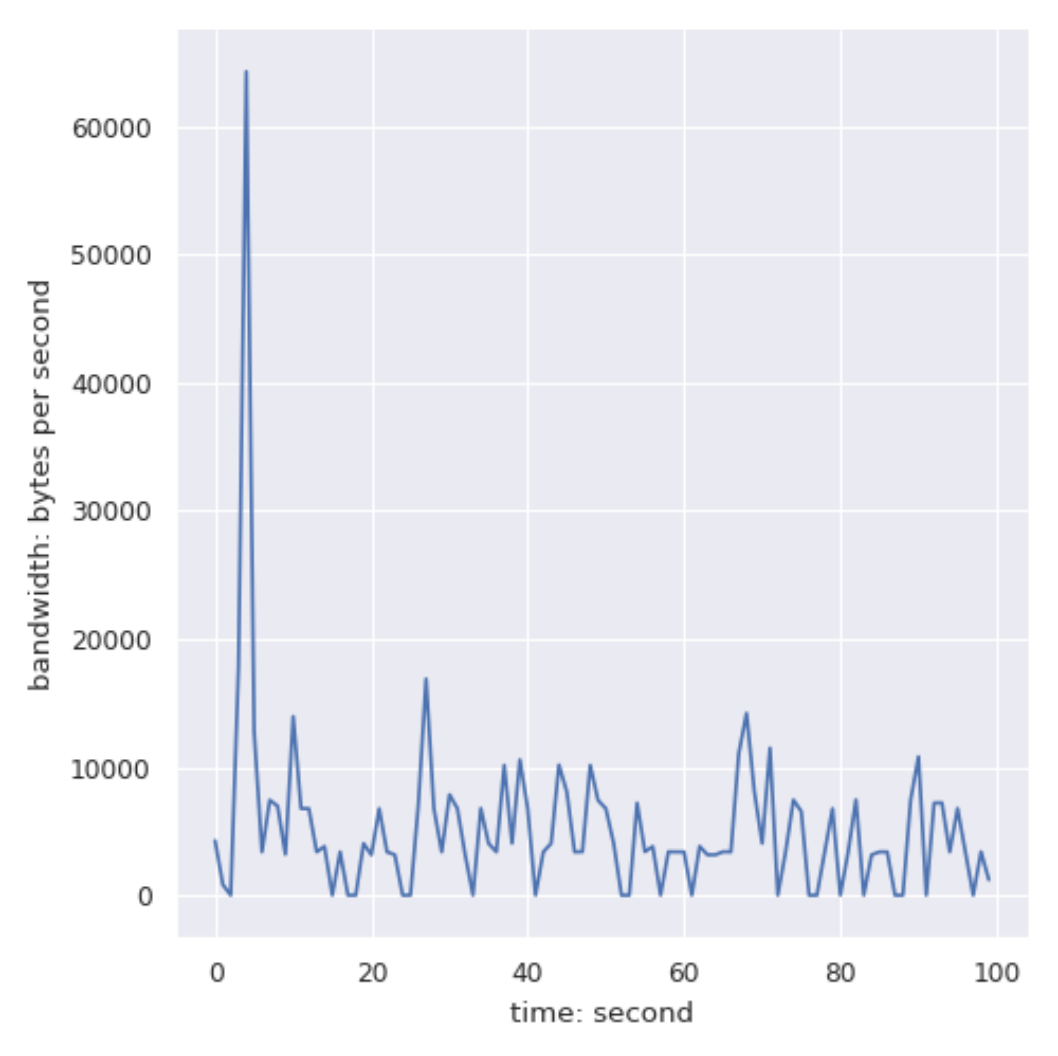
A


60000
50000
40000
30000
20000
10000
0
bandwidth: bytes per second
0
20
40
60
80
100
time: second

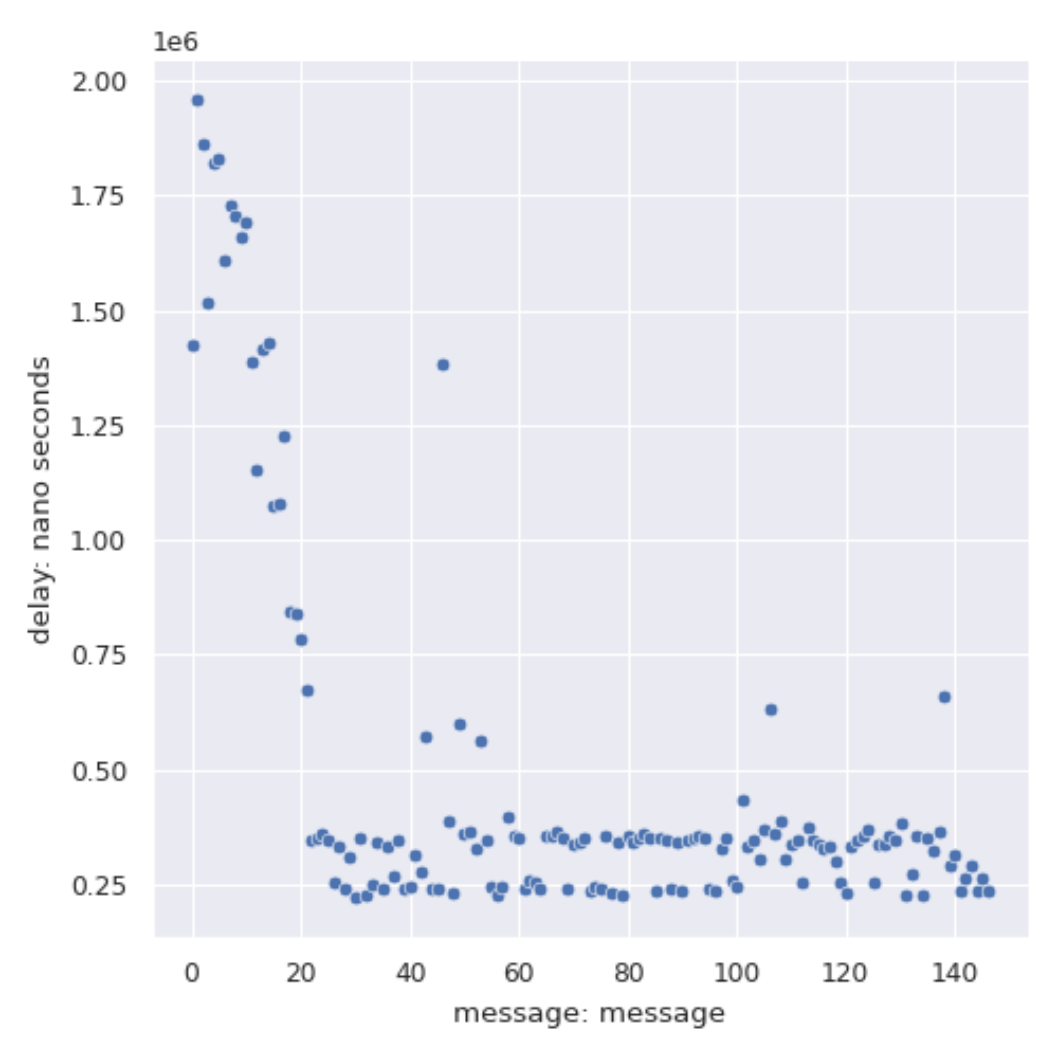
B


80000
60000
40000
20000
0
bandwidth: bytes per second
0
20
40
60
80
100
time: second

C

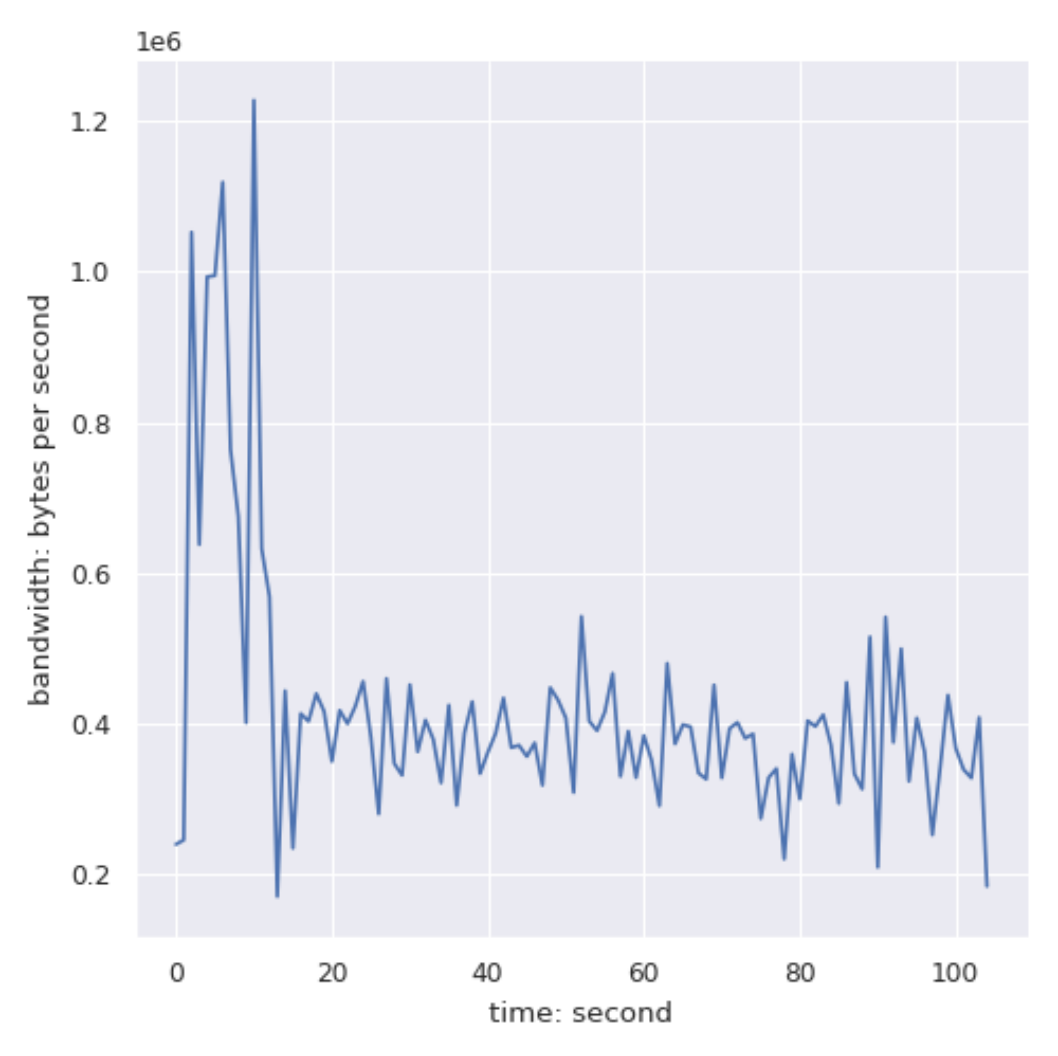

60000
50000
40000
30000
20000
10000
0
bandwidth: bytes per second
0
20
40
60
80
100
time: second

Delay

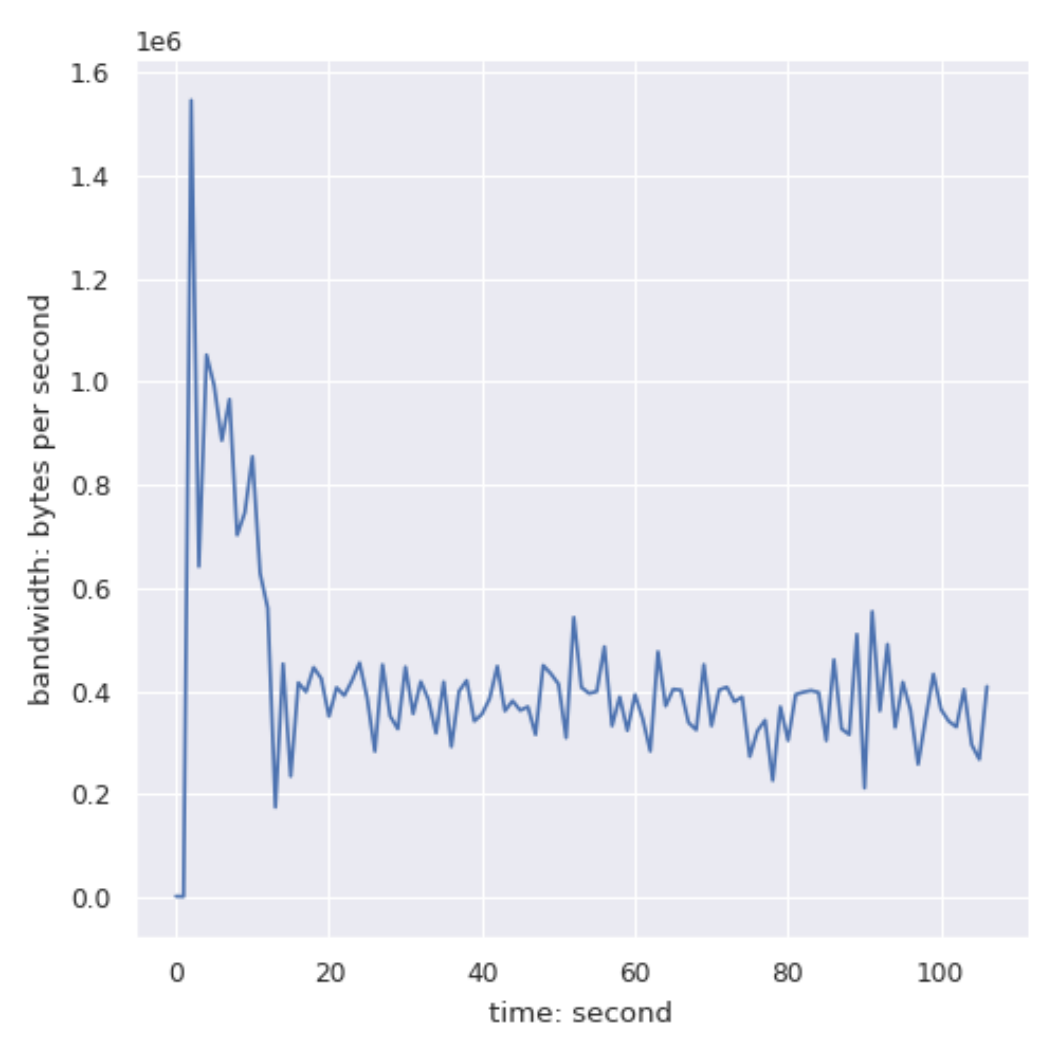


8 nodes, 5 Hz each, running for 100 seconds

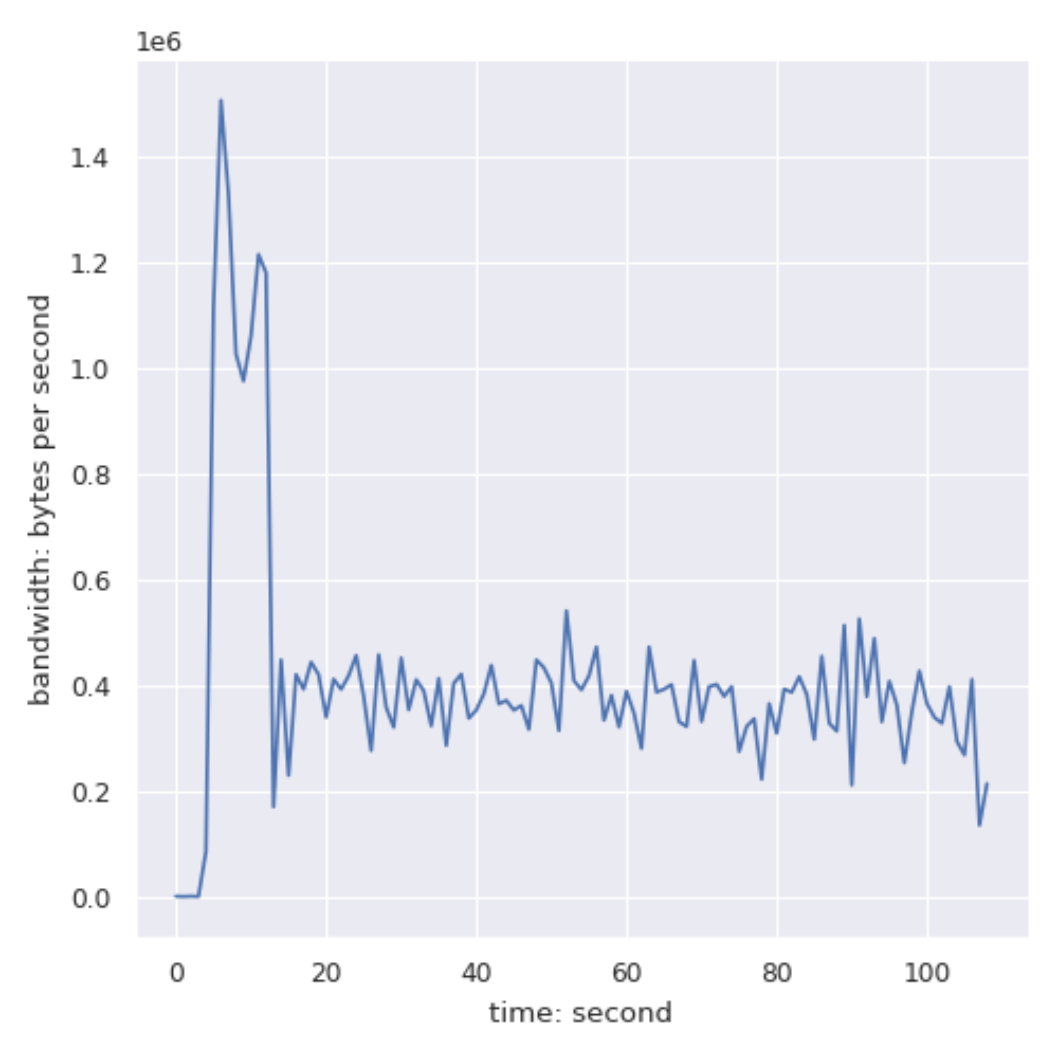
node1


1e6
1.2
1.0
0.8
0.6
0.4
0.2
bandwidth: bytes per second
0
20
40
60
80
100
time: second

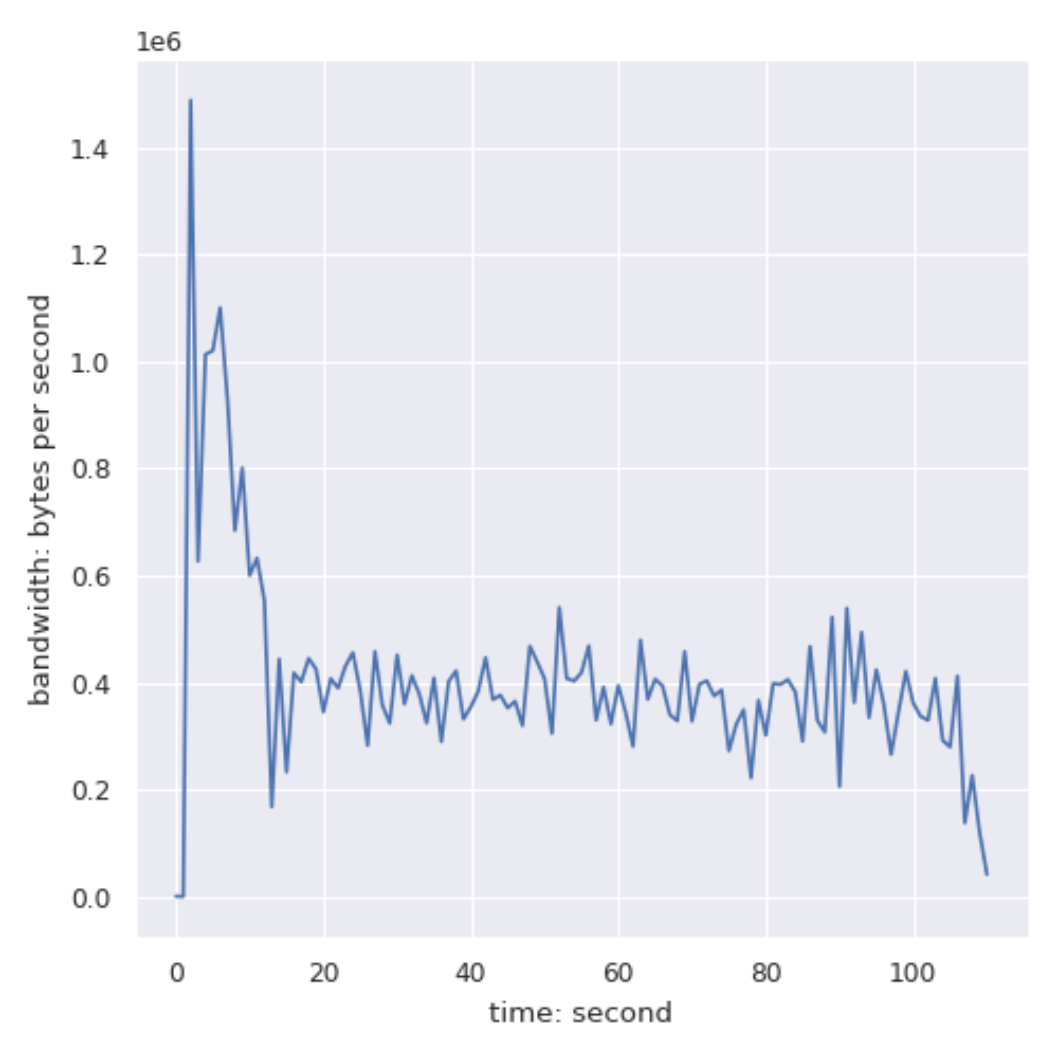
node2


1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
bandwidth: bytes per second
0
20
40
60
80
100
time: second

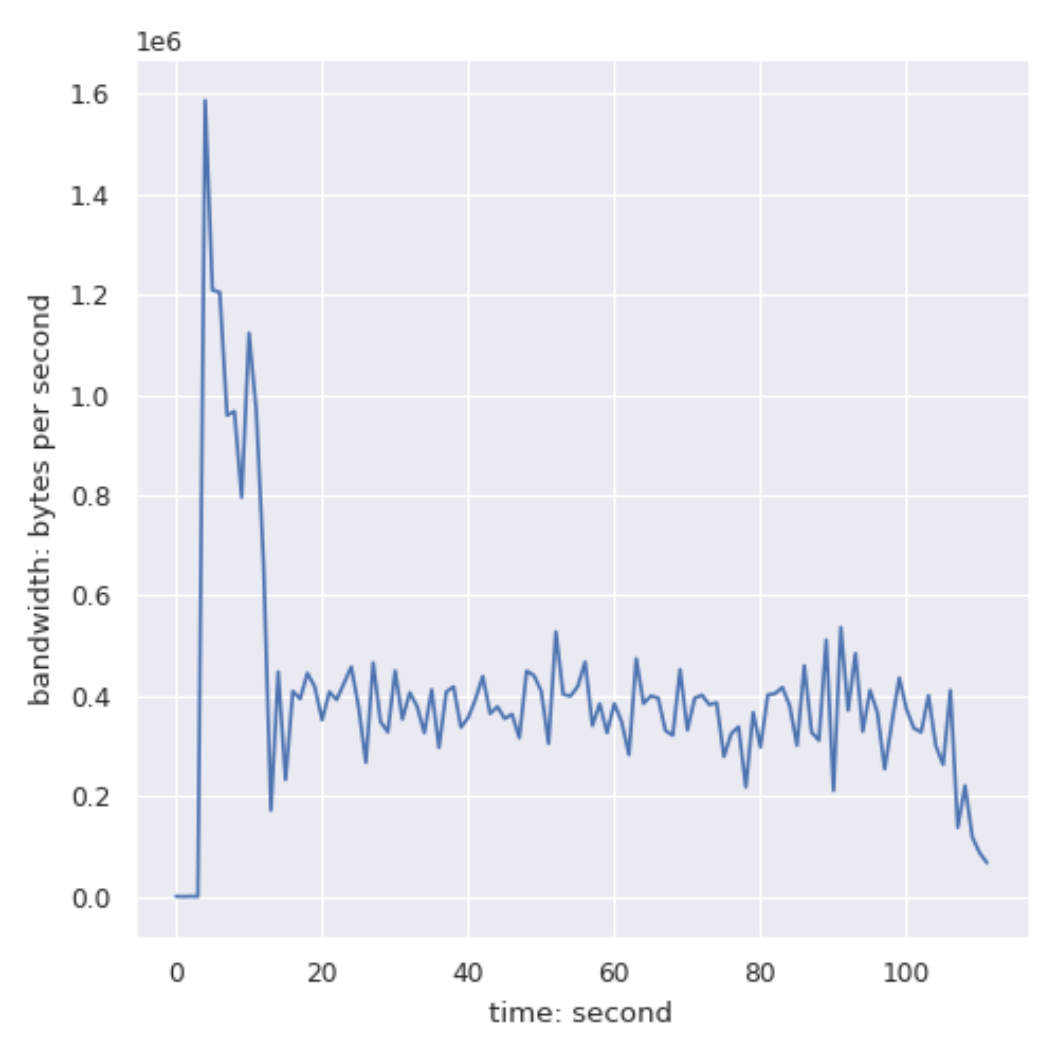
node3


1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
bandwidth: bytes per second
0
20
40
60
80
100
time: second

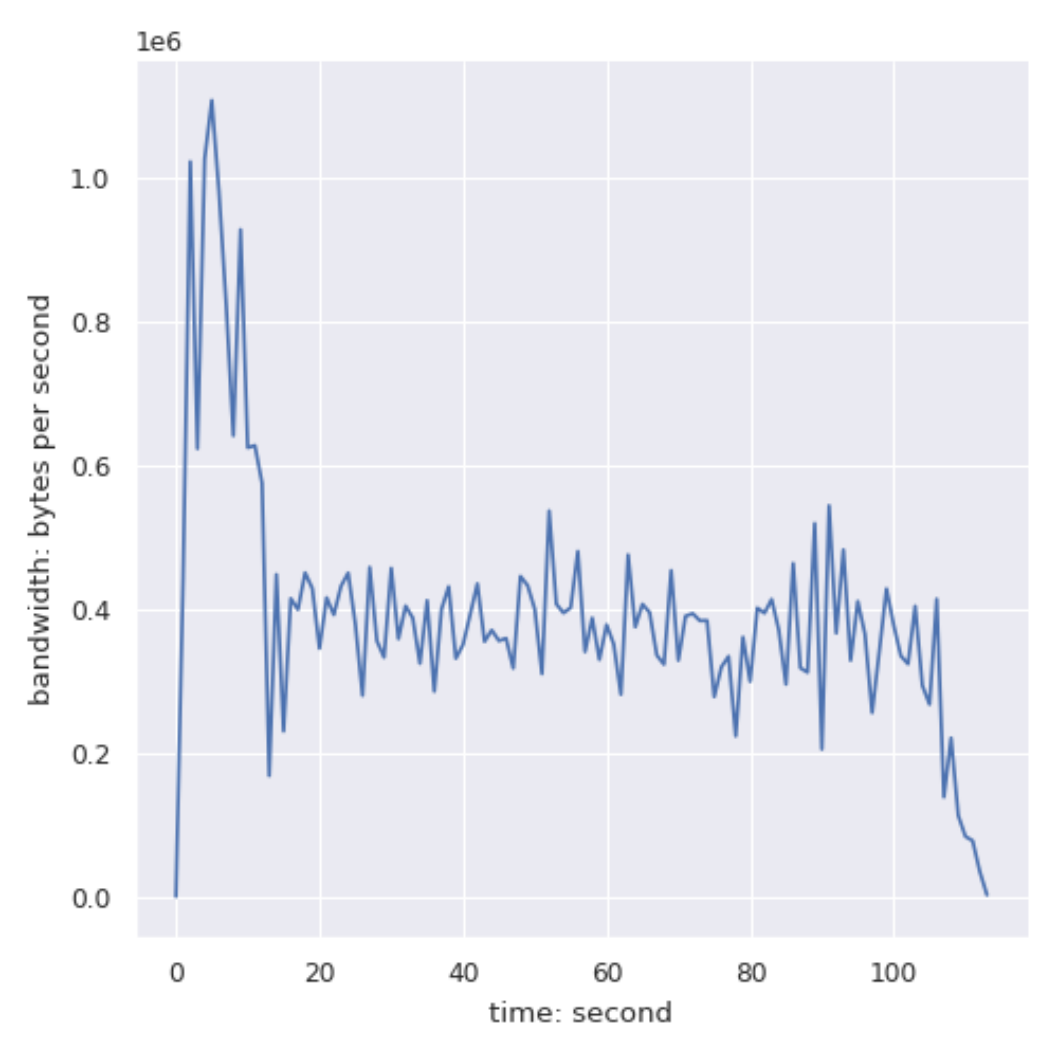
node4


1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
bandwidth: bytes per second
0
20
40
60
80
100
time: second

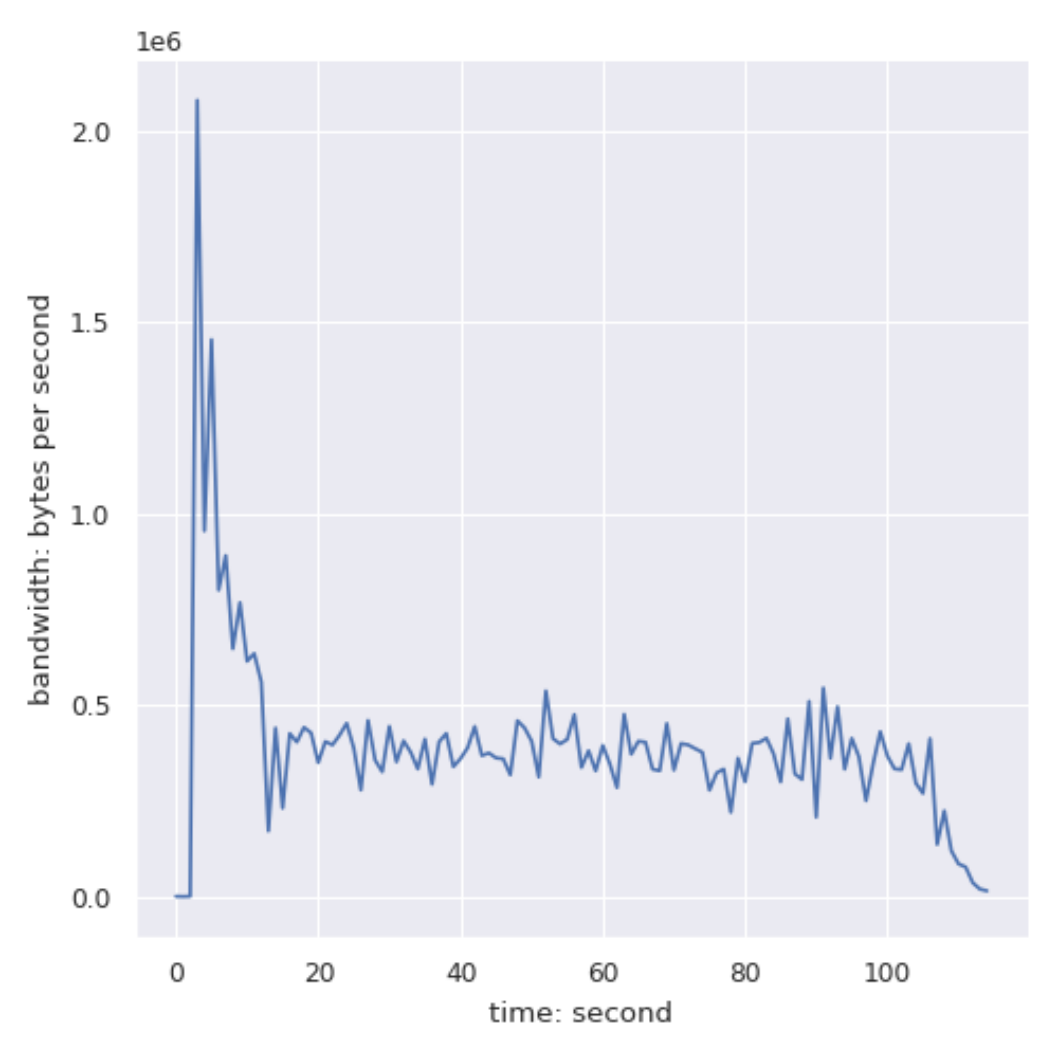
node5


1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
bandwidth: bytes per second
0
20
40
60
80
100
time: second

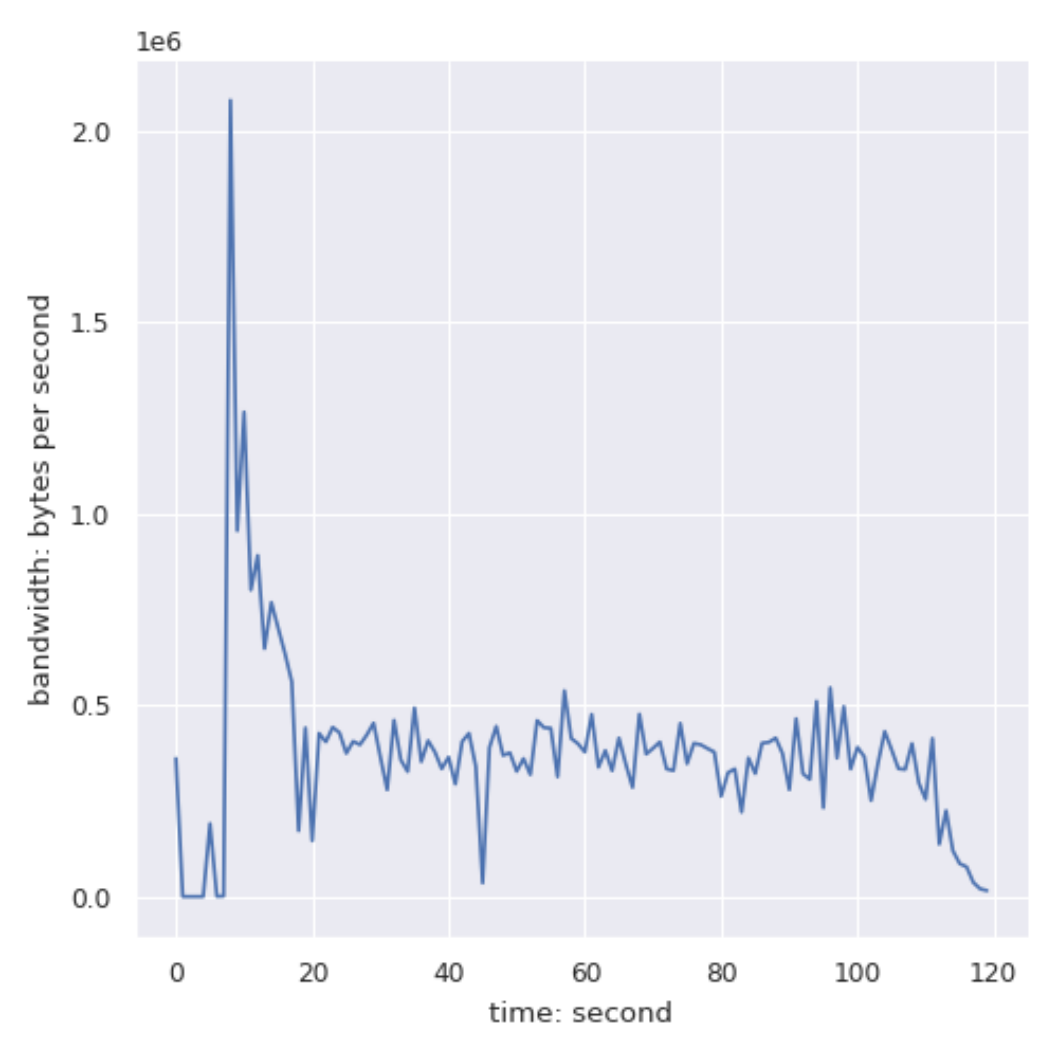
node6


1e6
1.0
0.8
0.6
0.4
0.2
0.0
bandwidth: bytes per second
0
20
40
60
80
100
time: second

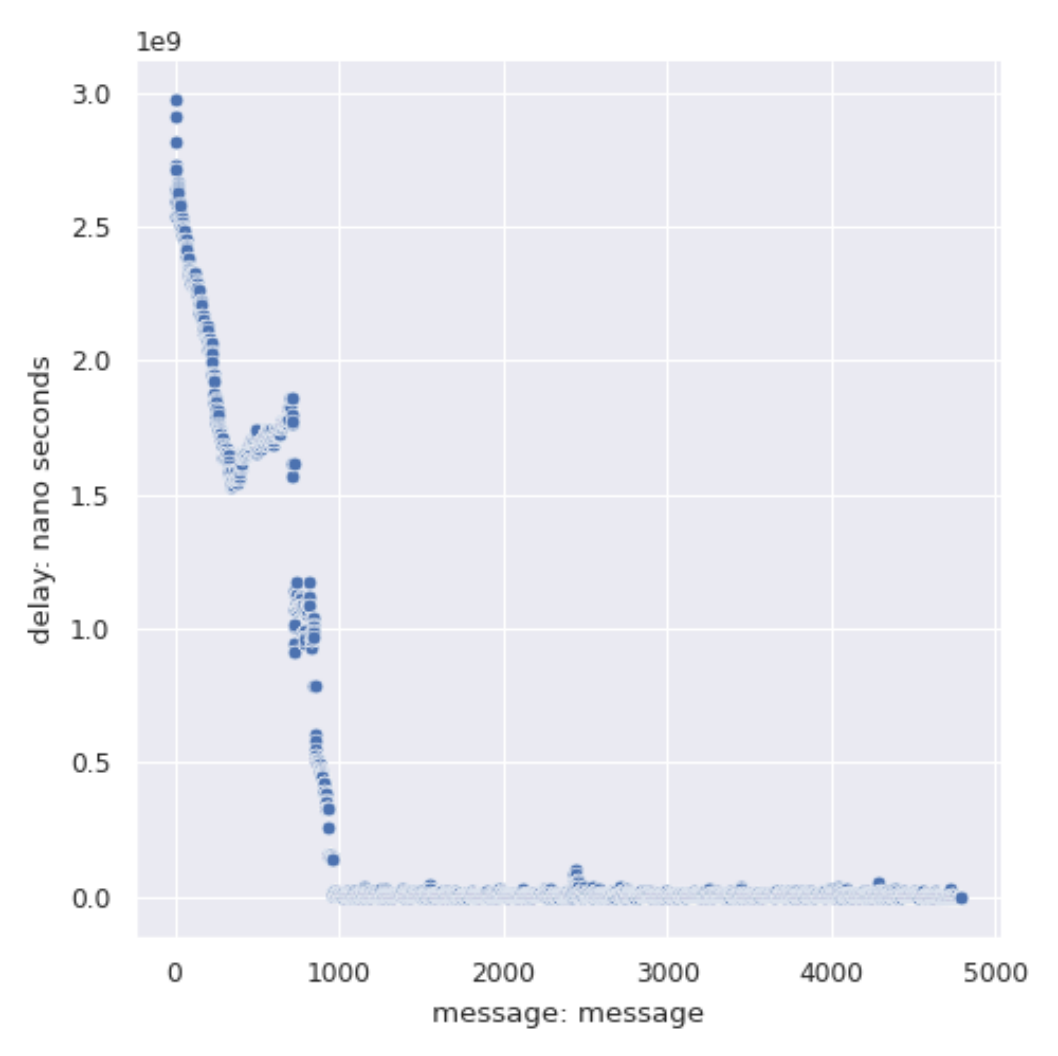
node7


1e6
2.0
1.5
1.0
0.5
0.0
bandwidth: bytes per second
0
20
40
60
80
100
time: second

node8

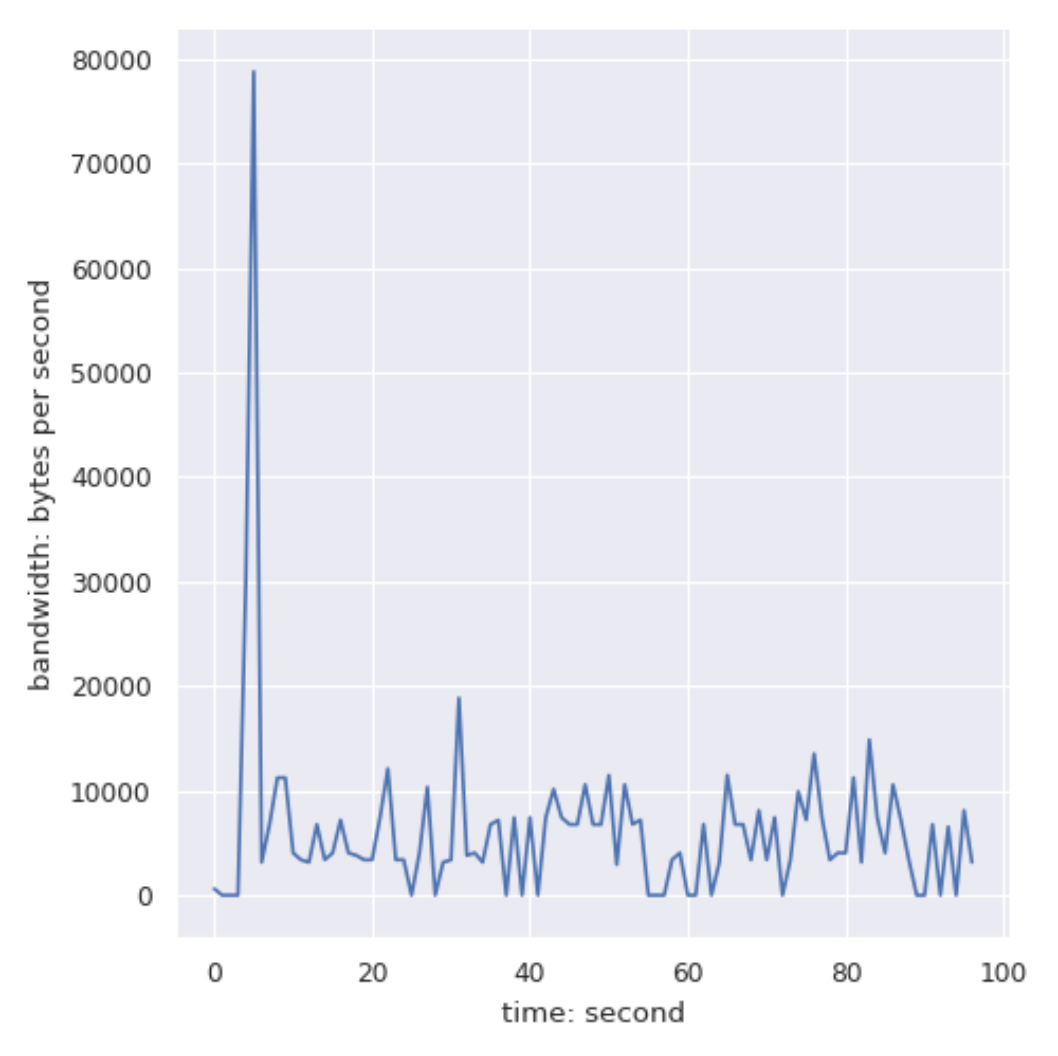

1e6
2.0
1.5
1.0
0.5
0.0
bandwidth: bytes per second
0
20
40
60
80
100
120
time: second

Delay

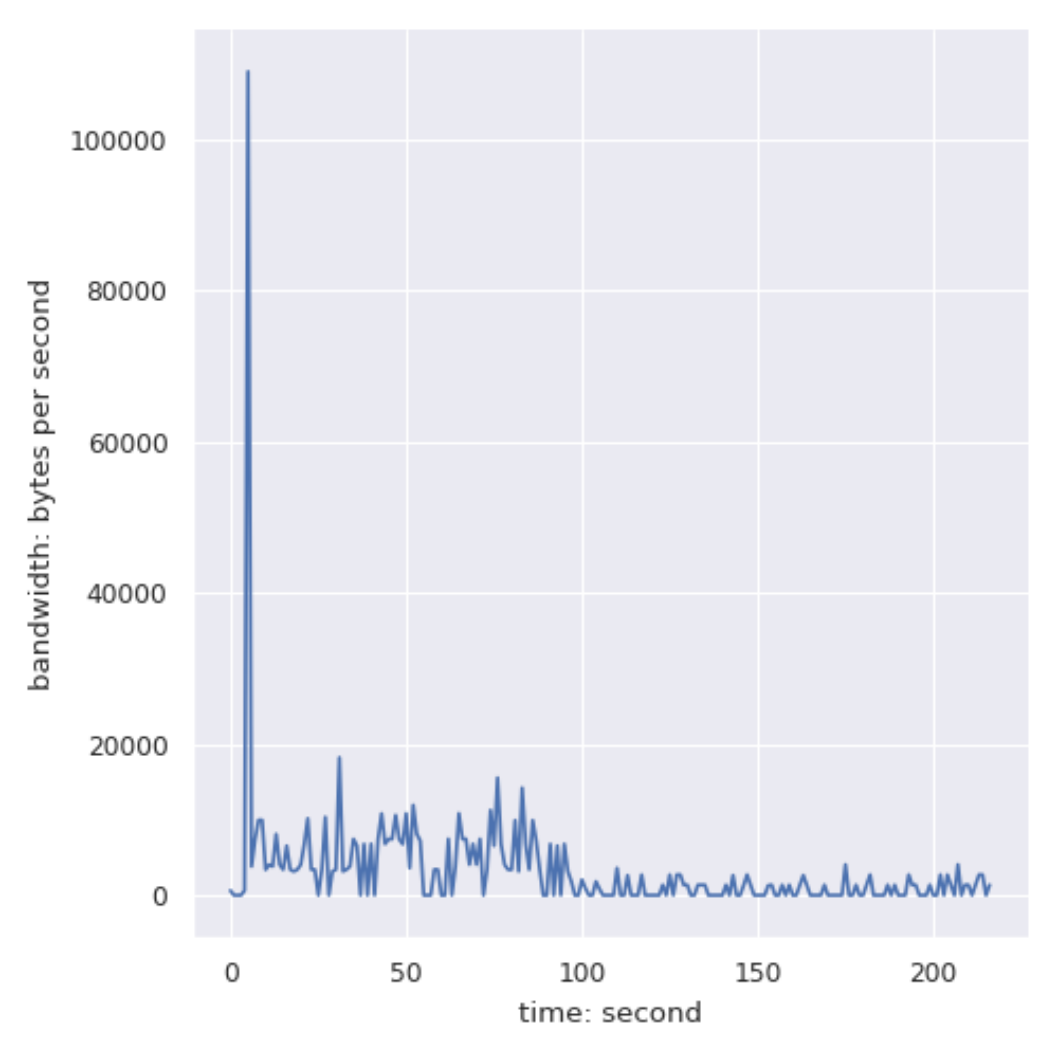


3 nodes, 0.5 Hz each, runing for 100 seconds, then one node fails, and the rest continue to run for 100 seconds

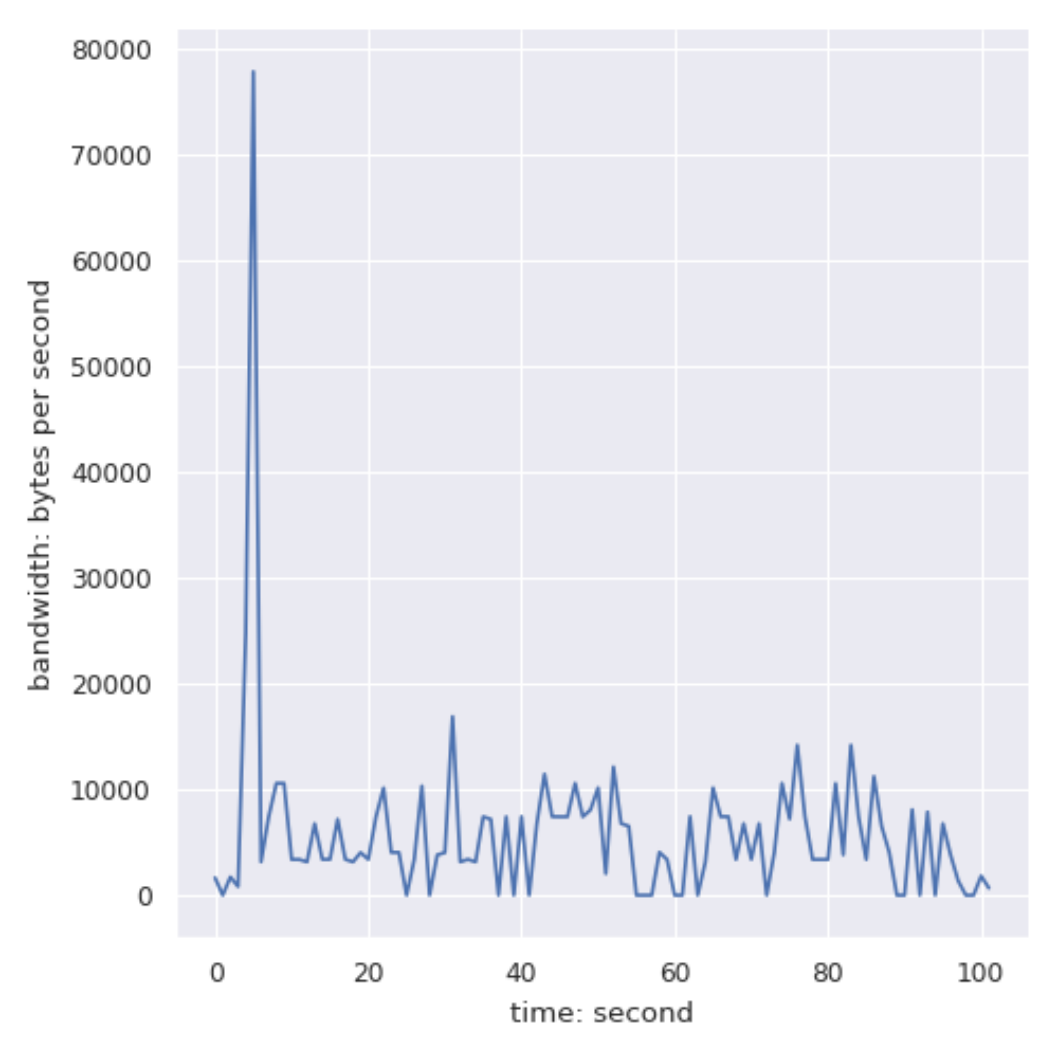
A


80000
70000
60000
50000
40000
30000
20000
10000
0
bandwidth: bytes per second
0
20
40
60
80
100
time: second

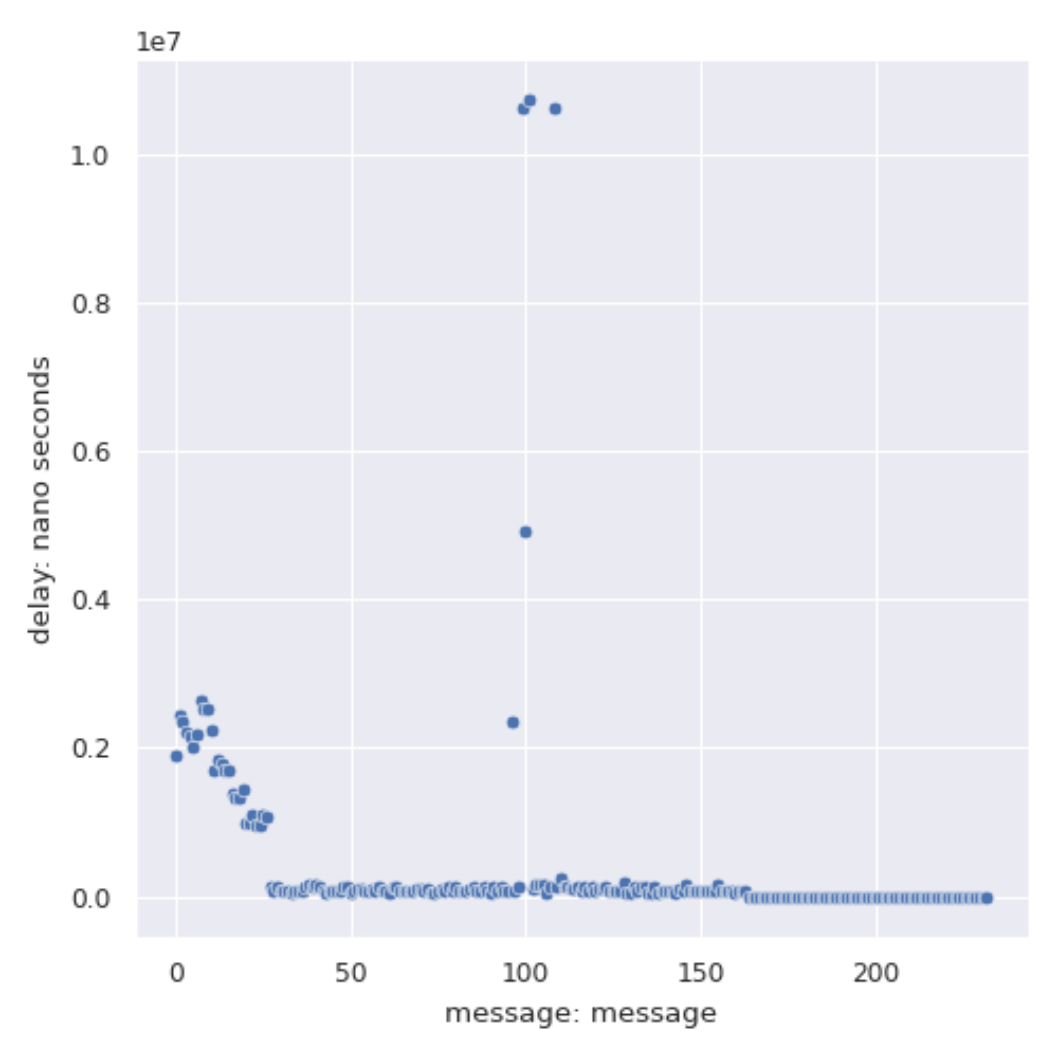
B


100000
80000
60000
40000
20000
0
bandwidth: bytes per second
0
50
100
150
200
time: second

C

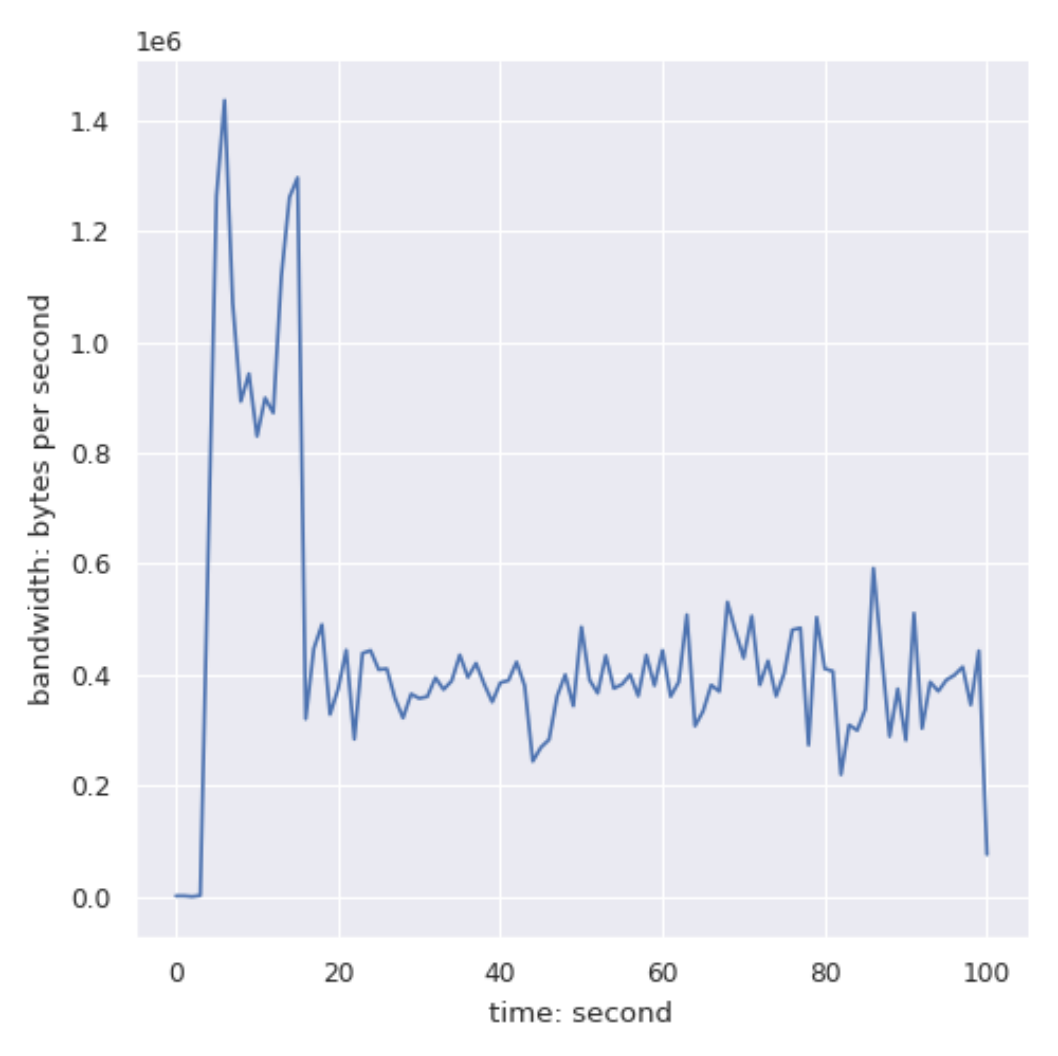

80000
70000
60000
50000
40000
30000
20000
10000
0
bandwidth: bytes per second
0
20
40
60
80
100
time: second

Delay

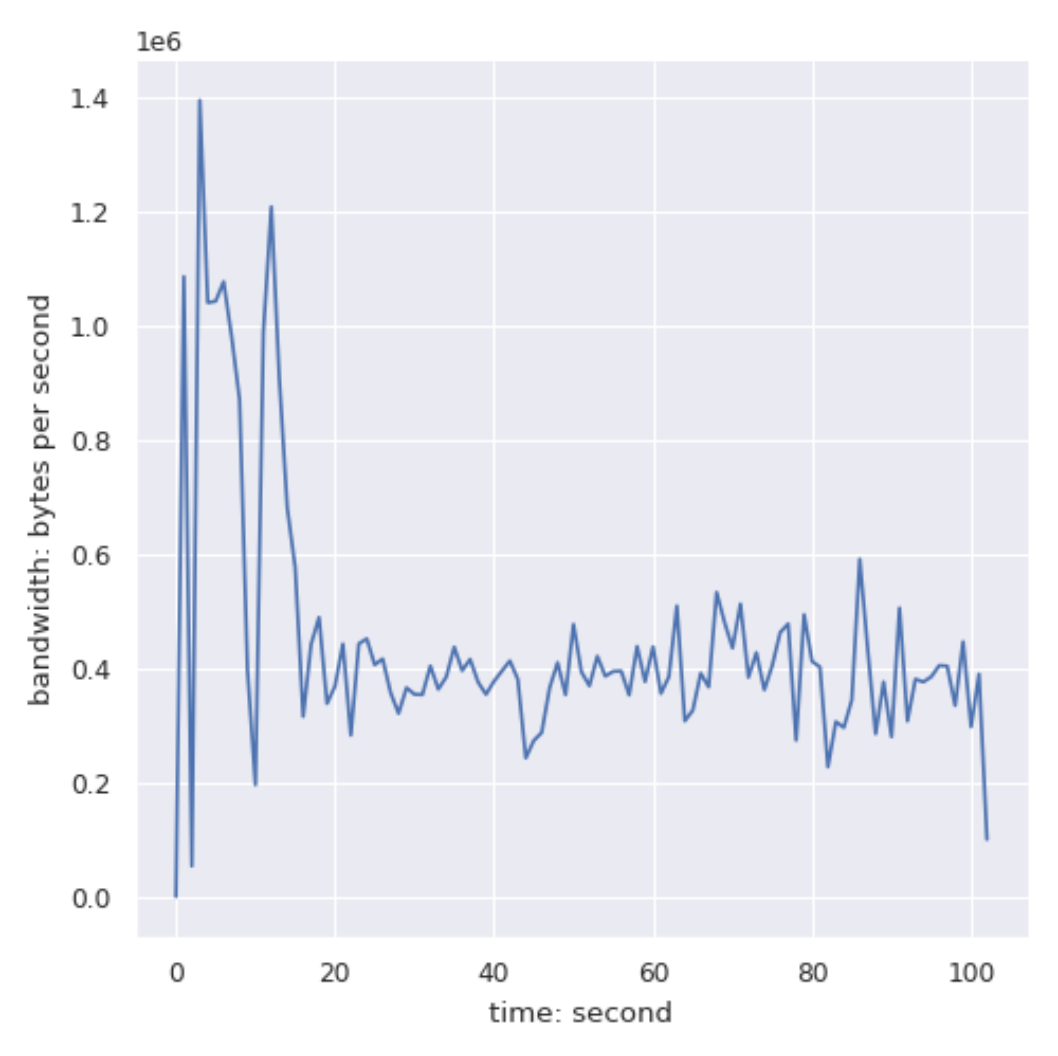


8 nodes, 5 Hz each, running for 100 seconds, then 3 nodes fail simultaneously, and the rest continue to run for 100 seconds

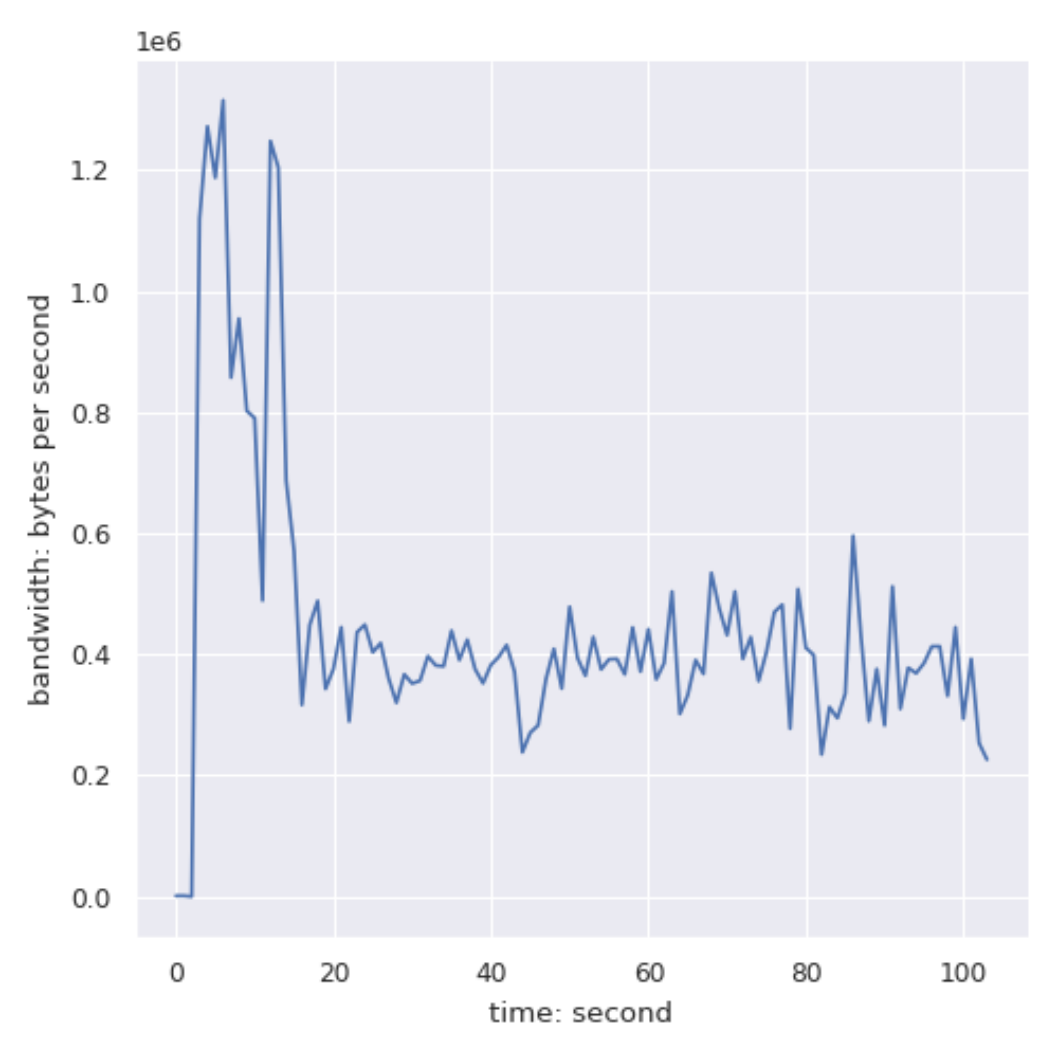
node1


1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
bandwidth: bytes per second
0
20
40
60
80
100
time: second

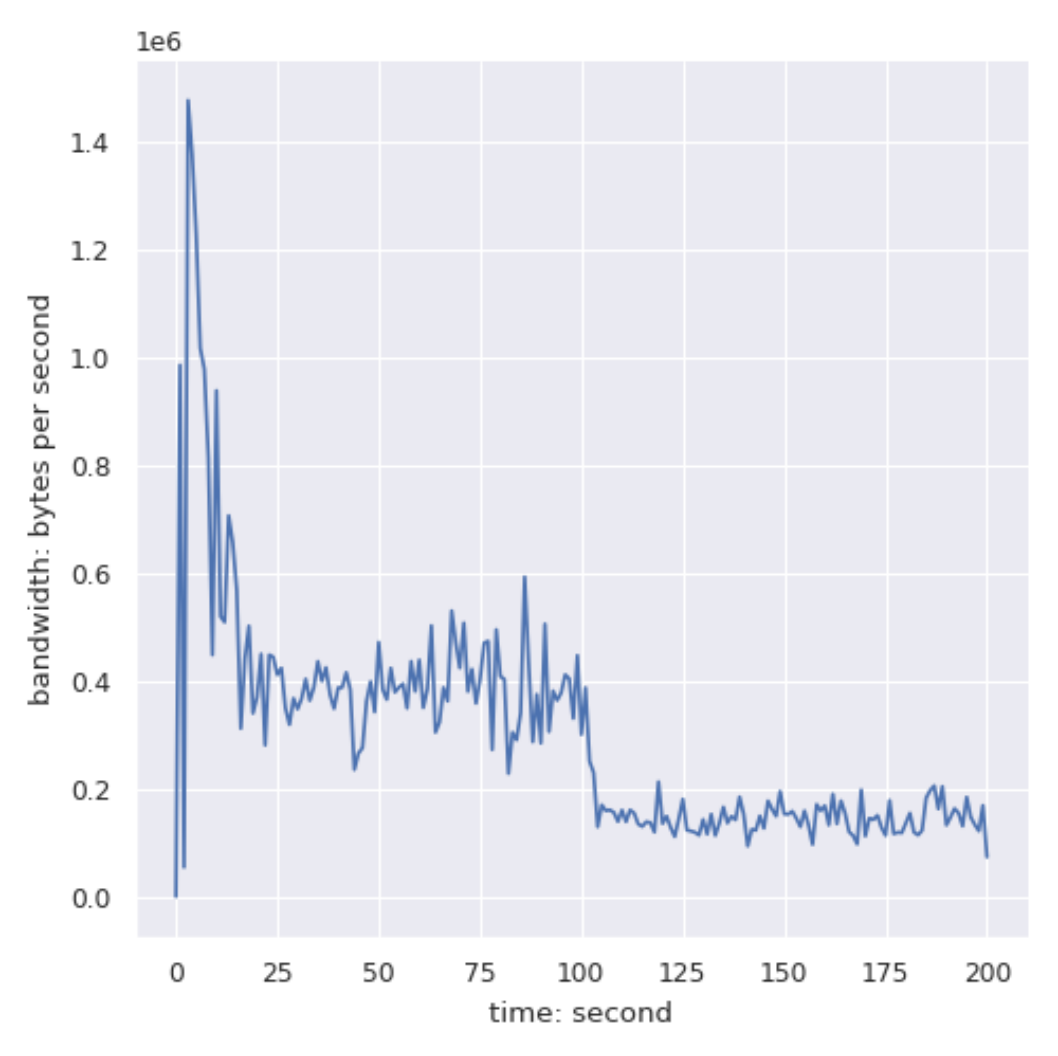
node2


1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
bandwidth: bytes per second
0
20
40
60
80
100
time: second

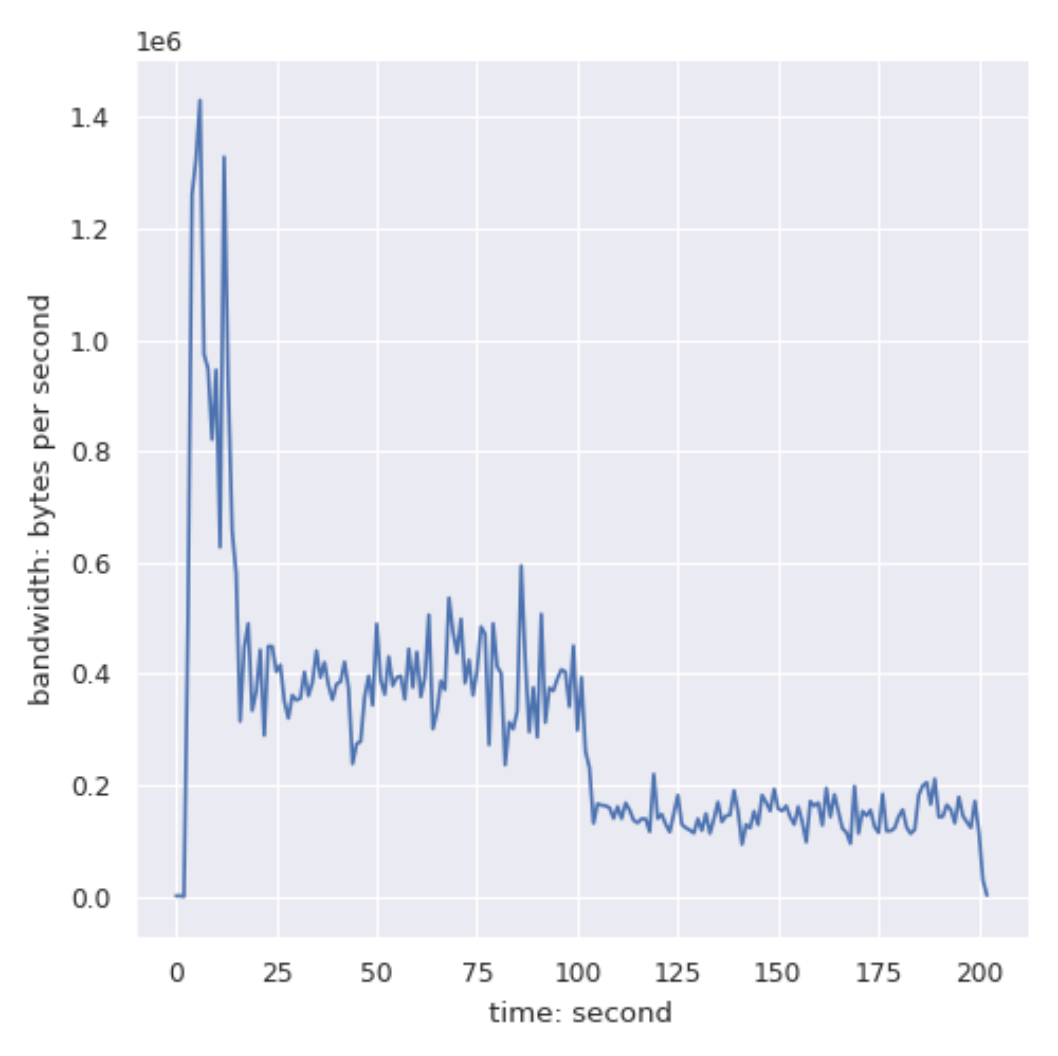
node3


1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
bandwidth: bytes per second
0
20
40
60
80
100
time: second

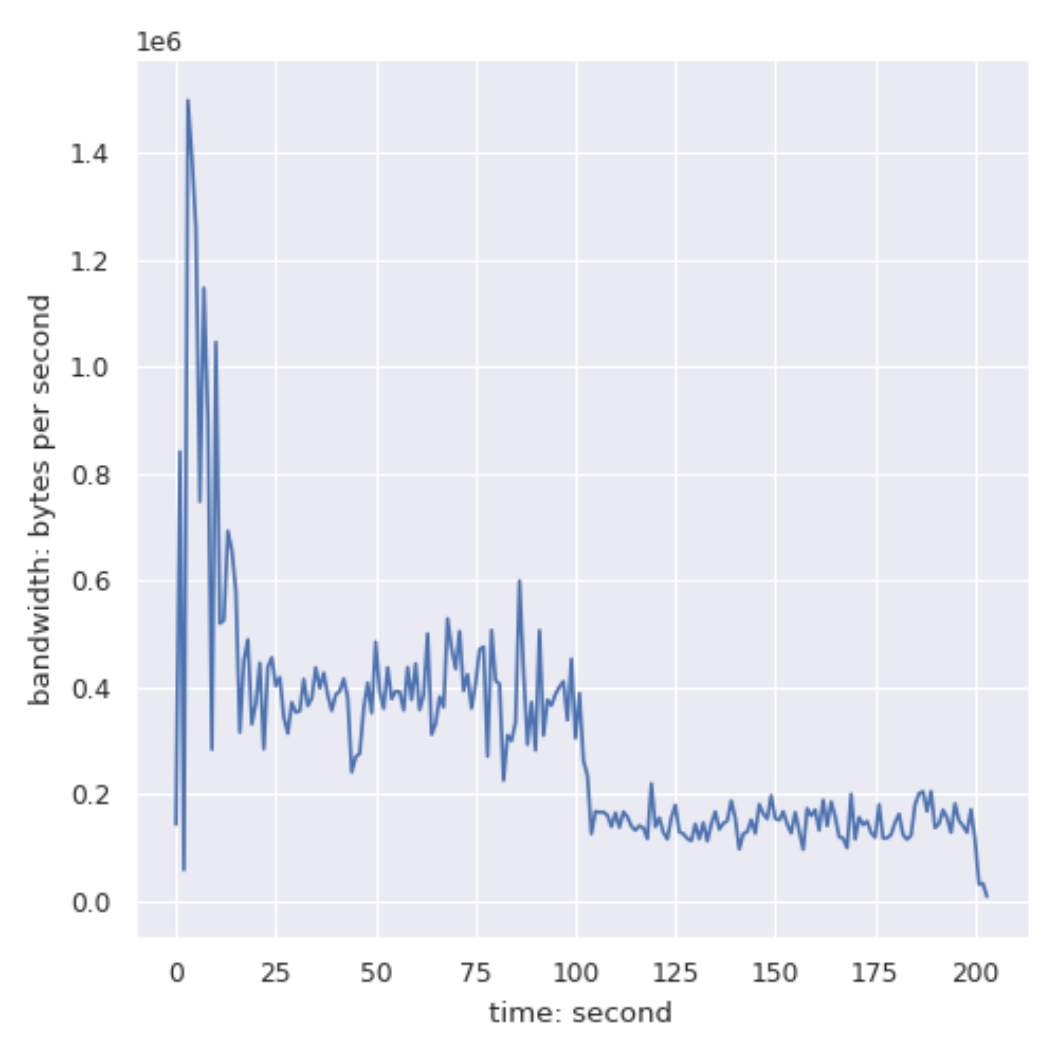
node4


1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
bandwidth: bytes per second
0
25
50
75
100
125
150
175
200
time: second

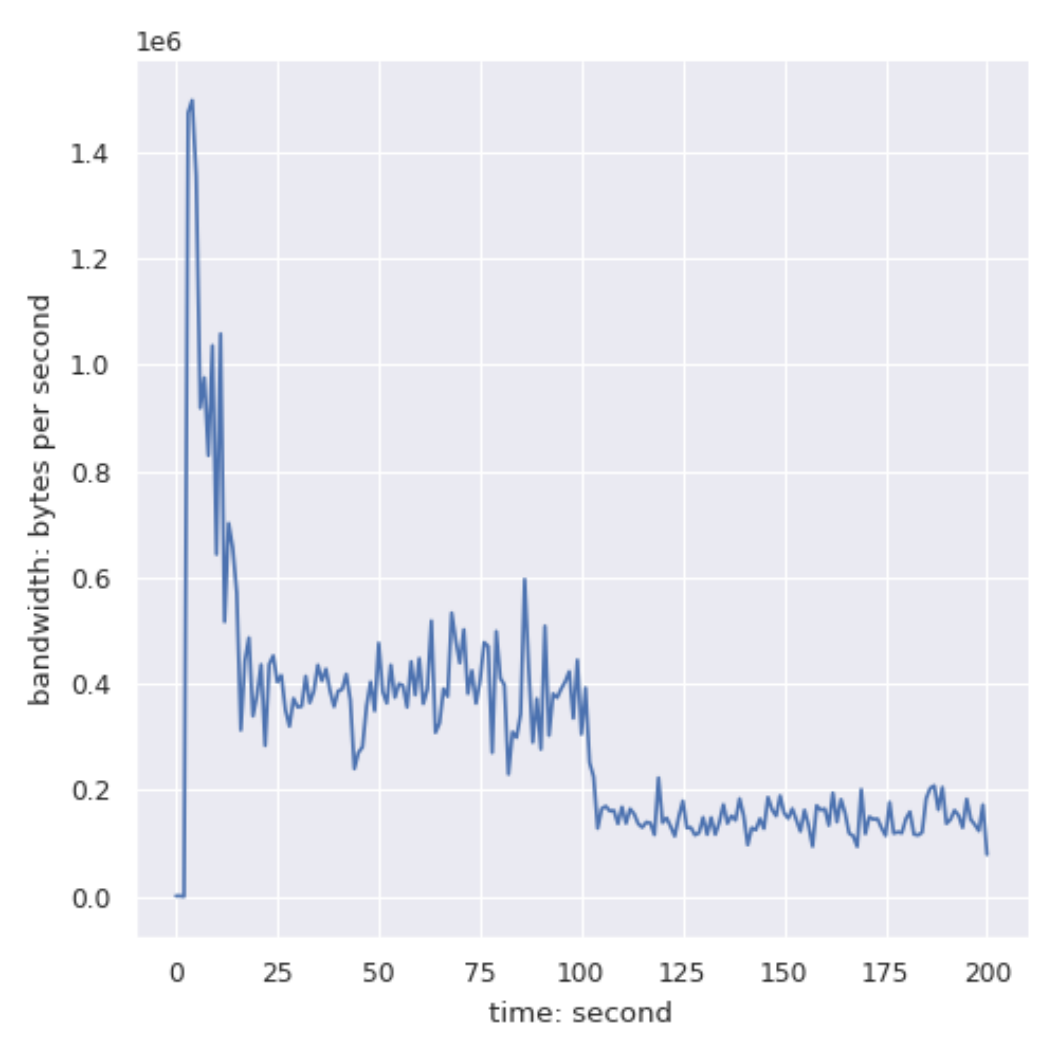
node5


1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
bandwidth: bytes per second
0
25
50
75
100
125
150
175
200
time: second

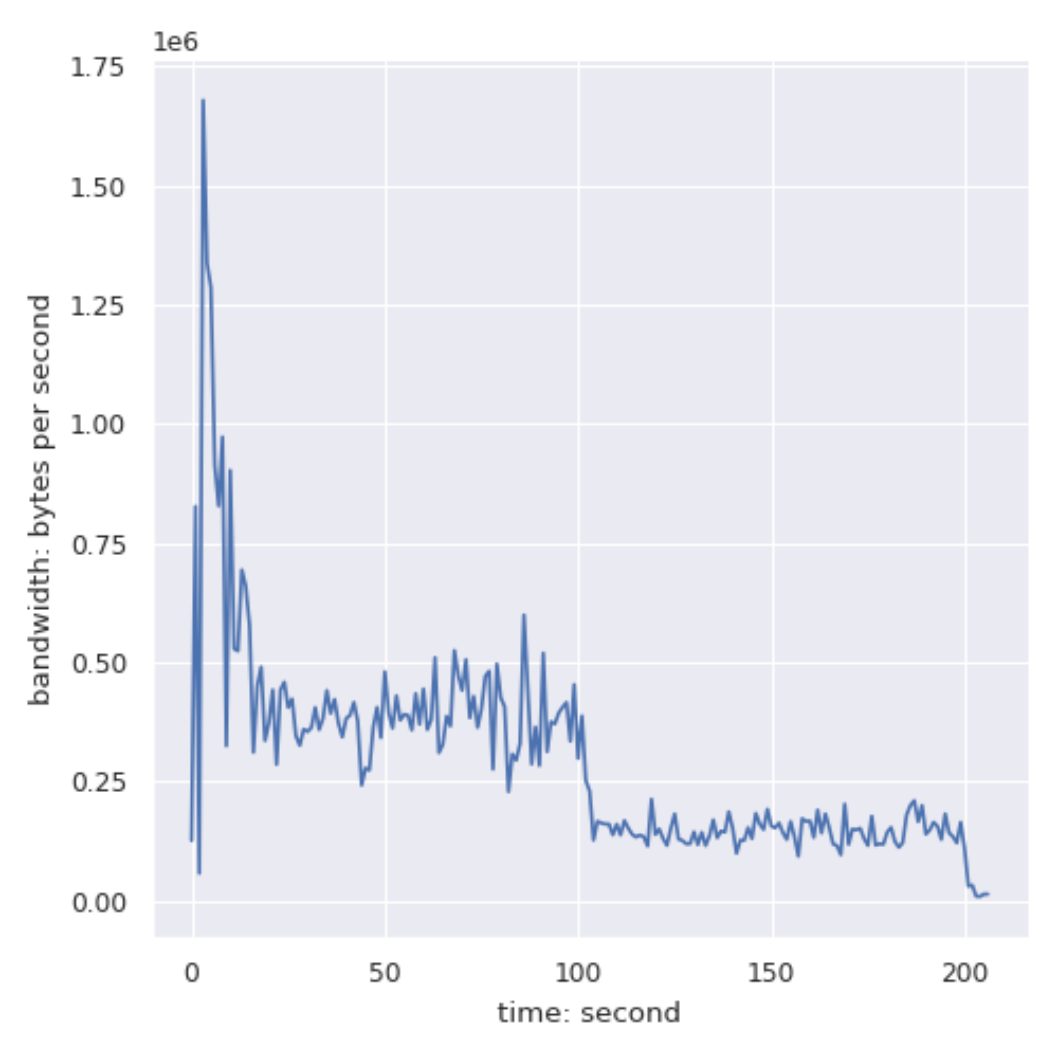
node6


1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
bandwidth: bytes per second
0
25
50
75
100
125
150
175
200
time: second

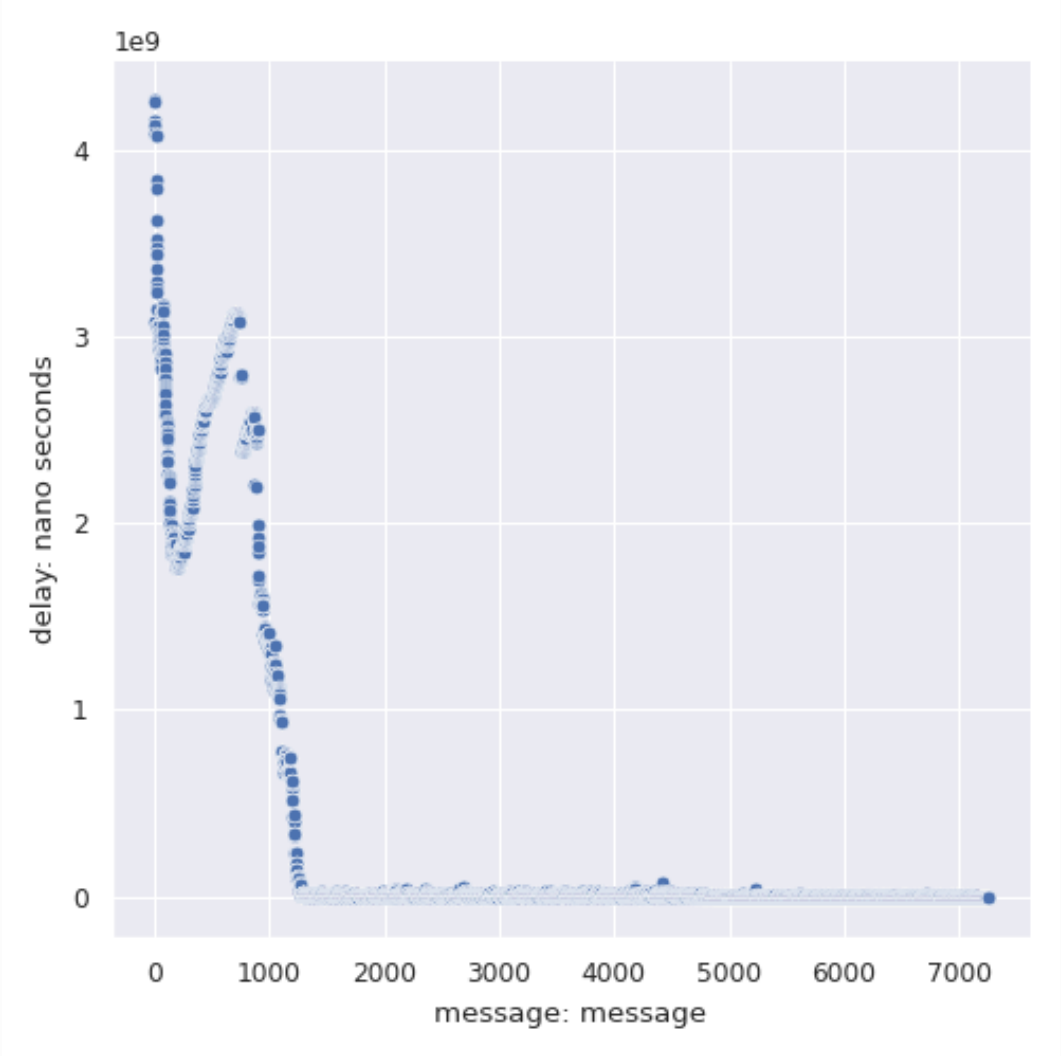
node7


1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
bandwidth: bytes per second
0
25
50
75
100
125
150
175
200
time: second

node8


1e6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
bandwidth: bytes per second
0
50
100
150
200
time: second

Delay


1e9
4
3
2
1
0
delay: nano seconds
0
1000
2000
3000
4000
5000
6000
7000
message: message