MP1 Report

Group Members

Name: Siyu Ren **NetID**: siyuren2

Name: Jiahan Chang **NetID**: jiahanc2 Cluster Number: g44

How to run our code

We have two .class file that works as the server thread and sender thread. To run the server program, run

```
$ cd mp1-nomoremp/draft/src
$ java ServerRun {node id} {port} {config file}
```

To run the sender program, please run:

```
$ cd mp1-nomoremp/draft/src
$ python3 -u gentx.py 0.5 | java SenderRun {node id} {port} {config file}
```

For each node, **the node id is {node1, node2, ..., node8}, port is {8081, 8082, ... ,8088}, config file path is {config1.txt, config2.txt, ..., config8.txt}.**

Explanation of design

We design one program to listen to the port and process received messages, and another program to read from python script and send multicast to nodes in the system.

How to ensure total ordering

We implemented ISIS algorithm to ensure total ordering. The message class here represents the messages being sent and delivered in the system. Below are some important property.

```
private String content; // what reads from the script
private String senderNodeId;
private String receiverNodeId;
private int MsgId;
private boolean deliverable; // to mark if the message is deliverable
private boolean isReturned; // to mark if the message is returned from other nodes with
prio
private int priorityId;
private String priority; // priorityId : senderNodeId, compare priorityId first then
compare senderNodeId
```

When the server receive the message, it will do different works due to the state of the message.

```
if (!message.isReturned()) { // receive the msg for the first time
    receiveFirst(message);
} else if (message.isReturned() && !message.isDeliverable()) { // receive the back
message
    receiveBack(message);
} else if (message.isDeliverable()) { // receive the final message
    receiveFinal(message);
}
```

- When the server receives the message for the first time, it wil set the priority of message and save the message in a map. Then it will send the message back to the sender and update its priority.

```
public void receiveFirst(Message message) {
    // priority id ++
    int priorityId = this.getPriorityId() + 1;
    this.setPriorityId(priorityId);
    message.setDeliverable(false);
    message.setPriority(priorityId + ":" + this.getNodeName());
    message.setPriorityId(priorityId);
    message.setReturned(true);
    // save the message
    this.getMessages().put(message.getSenderNodeId() + ":" + message.getMsgId(),
message);
    // send back the msg to sender
    sendBack(message);
}
```

- When the server receives the returned message from other servers, put the msg in a map. If it receives msgs from all servers, it will sort the msg by priority and get the largest msg and multicast to other nodes

```
public void receiveBack(Message message) {
    if (!getReplyMsgs().containsKey(message.getMsgId()))
getReplyMsgs().put(message.getMsgId(), new ArrayList<>());
    List<Message> messages = getReplyMsgs().get(message.getMsgId());
    messages.add(message);
    for (Integer msgId : getReplyMsgs().keySet()) {
        List<Message> messageList = getReplyMsgs().get(msgId);
        // if collect all the return message
        if (messageList.size() >= this.getGroup().size()) {
            sortMessages(messageList);

            // get max msg
            Message maxMsg = messageList.get(messageList.size() - 1);
            maxMsg.setDeliverable(true);
            // remove the msg
            this.getReplyMsgs().remove(msgId);
            // multicast to other nodes
```

```
                sendFinal(maxMsg);
            } else {
                getReplyMsgs().put(message.getMsgId(), messageList);
            }
        }
    }
```

- When the server receives the final msg, it will remove the message with the same sender and msgId. Then add the final msg to messages and sort the messageList. if the first msg is deliverable, deliver it and remove it from messages.

```
public void receiveFinal(Message message) {
    // update the msg
    this.getMessages().remove(message.getSenderNodeId() + ":" + message.getMsgId());
    this.getMessages().put(message.getSenderNodeId() + ":" + message.getMsgId(),
message);
    if (message.getPriorityId() > this.getPriorityId()) {
        this.setPriorityId(message.getPriorityId());
    }
    List<Message> messages = new ArrayList<>(this.getMessages().values());
    // sort the received msgs
    sortMessages(messages);
    // if the smallest msg is deliverable deliver it and remove it
    if (messages.get(0).isDeliverable()) {
        Message firstMsg = messages.remove(0);
        deliver(firstMsg);
        this.getMessages().remove(firstMsg.getSenderNodeId() + ":" +
firstMsg.getMsgId());
    }
}
```

How to deal with failure

We use TCP connection, so we can know if a node has failed after a TCP timeout. We user the sender to detect failure. When the sender multicast a message to all nodes, if it cannot connect to the node with the timeout, it will multicast a message that imforms other nodes that the node has failed.

```
this.getGroup().remove(message.getReceiverNodeId());
multiCastFailure(message.getReceiverNodeId());
```

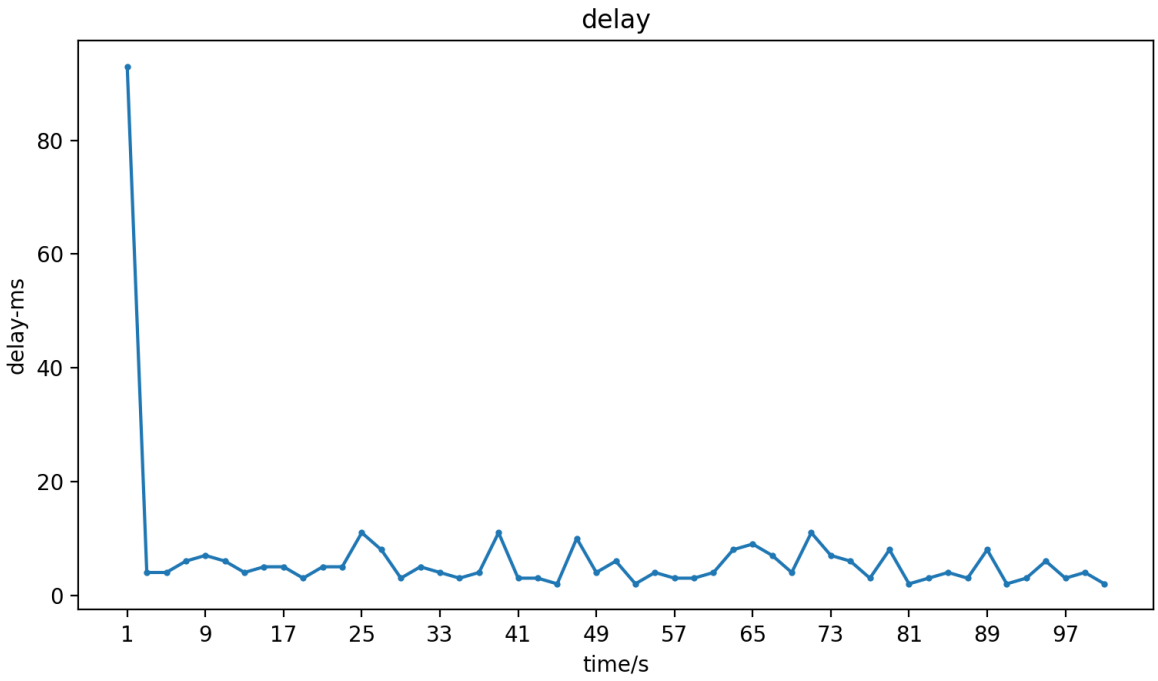
Then the other nodes will remove the node from the groups and remove the messages sent from the node.

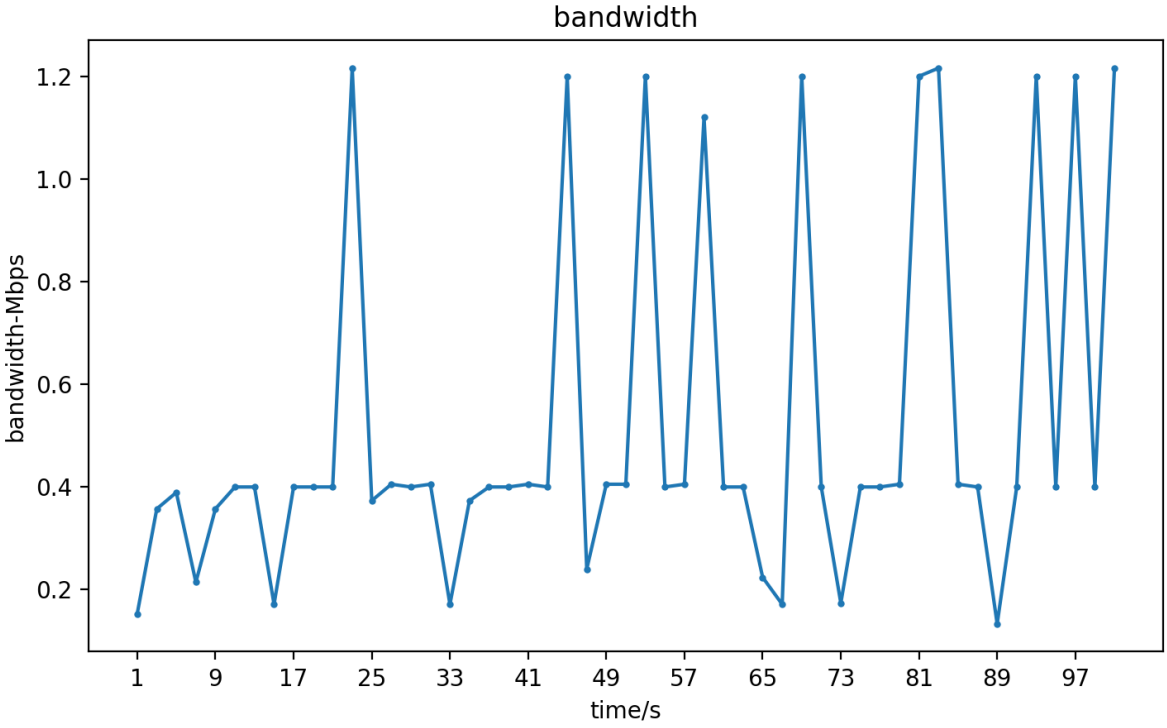
```
private void deleteNode(String nodeId) {
    this.getGroup().remove(nodeId);
    for (Integer msgId : this.getReplyMsgs().keySet()) {
        List<Message> messages = getReplyMsgs().get(msgId);
        messages.removeIf(msg -> msg.getReceiverNodeId().equals(nodeId));
    }
    for (String key : this.getMessages().keySet()) {
        if (getMessages().get(key).getSenderNodeId().equals(nodeId)) {
            getMessages().remove(key);
        }
    }
}
```

After that, the server nodes will continue to work.

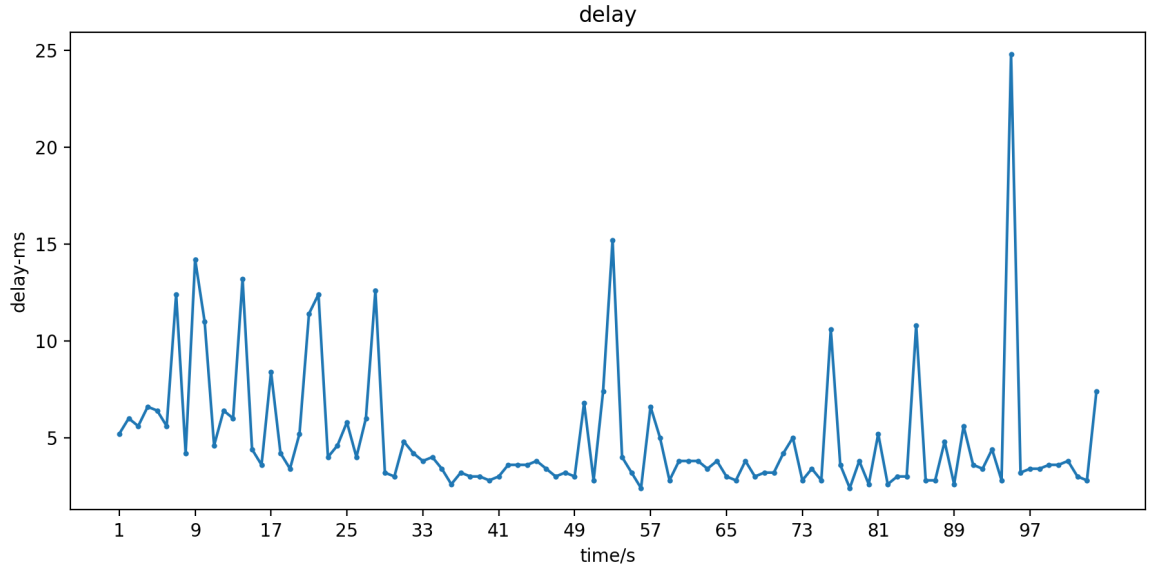
Graph of the evaluation

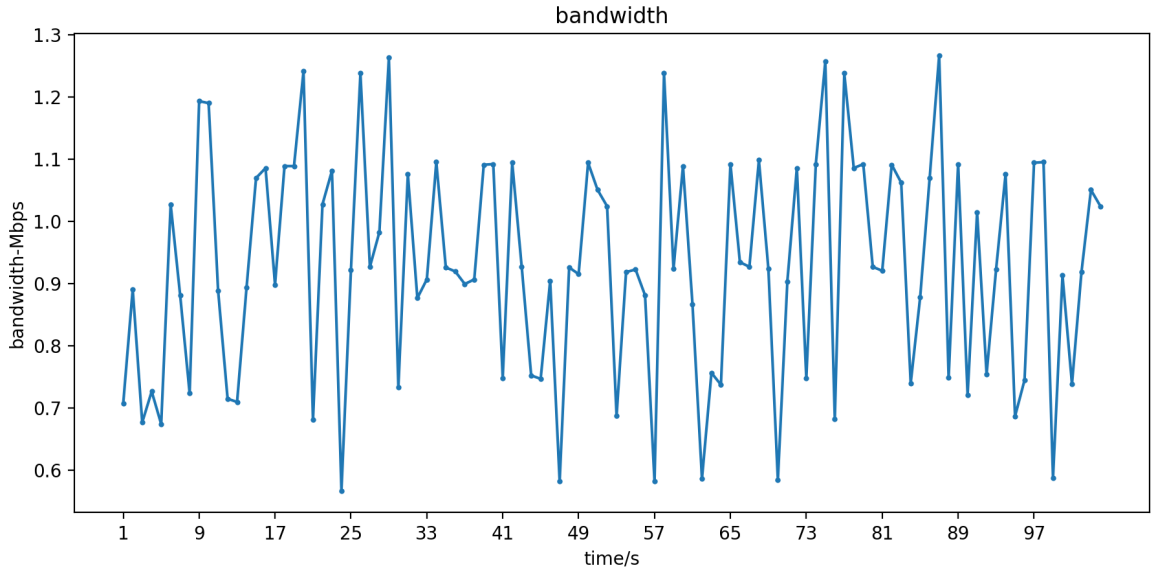
3 nodes, 0.5 Hz each, running for 100 seconds



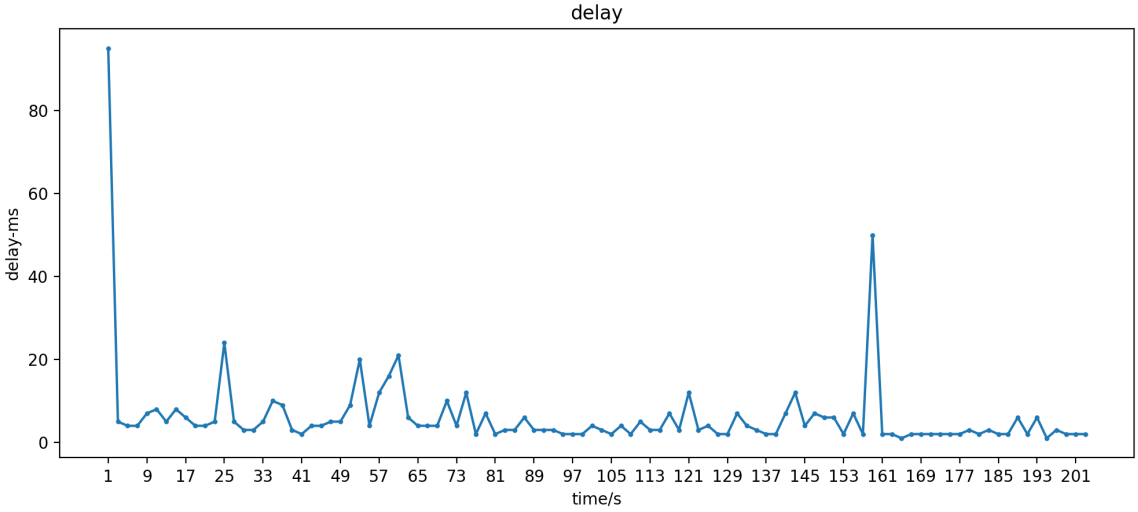


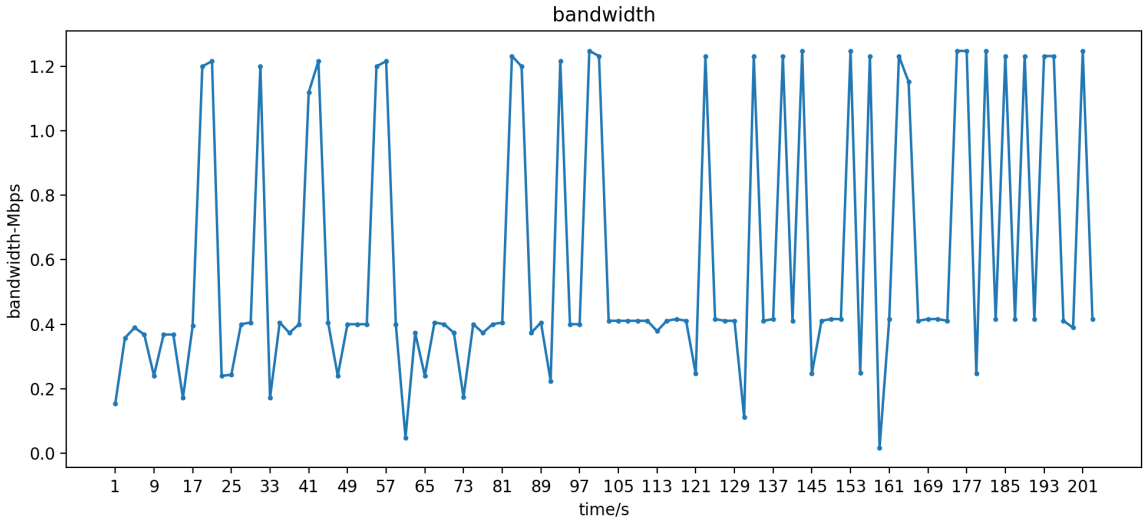
8 nodes, 5 Hz each, running for 100 seconds



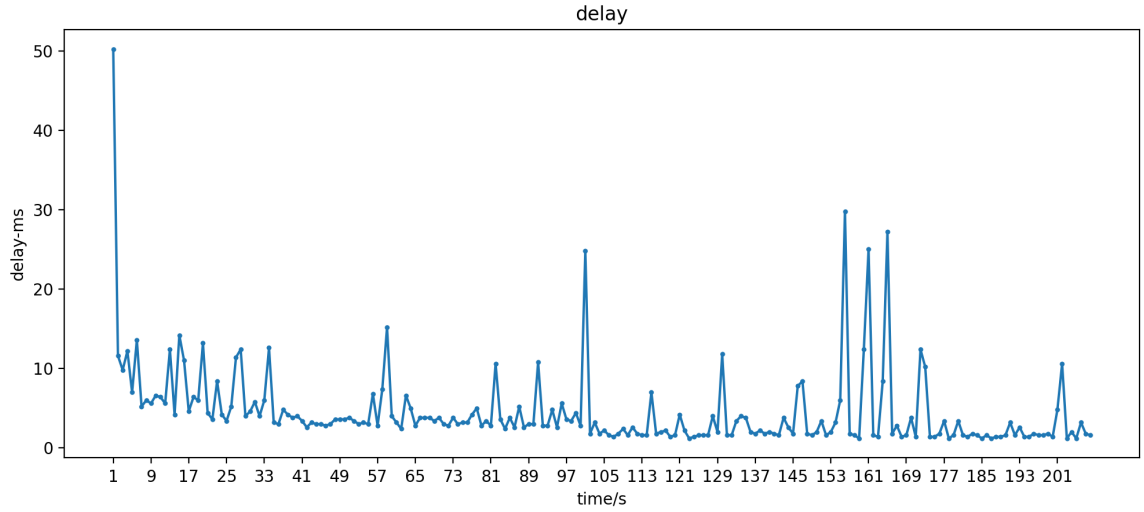


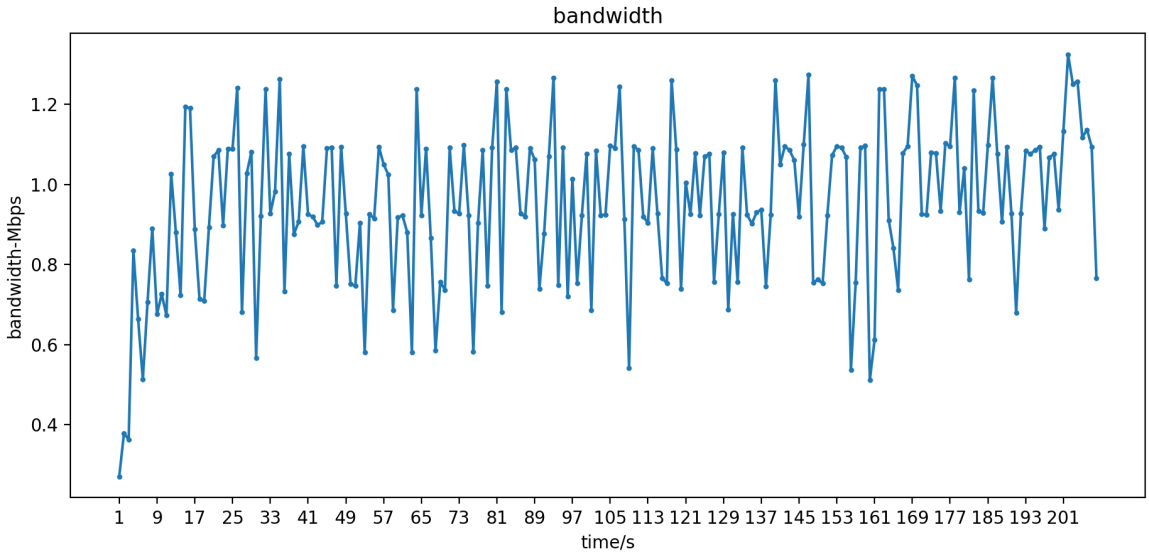
3 nodes, 0.5 Hz each, runing for 100 seconds, then one node fails, and the rest continue to run for 100 seconds





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





Summary of the graphs

From the figures, we can find that no matter whether there is break or not, how many nodes are, the bandwidth and the delay always keep in the same range. This shows that our program has successfully achievd total odering.