MP1 Report

Yang Zhou 3190110354 yang.19@intl.zju.edu.cn
Zheyu Fu 3190110355 zheyu.19@intl.zju.edu.cn

1. Protocol design

In order to comply with the requirements of this MP, we must develop a specialized ISIS algorithm that incorporates R-multicast functionality. The algorithm must ensure that any incoming message is checked for a previous encounter with the same (ID, priority, sender) combination. If the message has been previously received, it must be discarded to ensure that R-multicast implementation is effective. On the other hand, if the message is new, the node must disseminate it to all other group nodes without making any modifications. If the incoming message contains a proposal priority, the node must handle it appropriately. However, if the message does not contain a proposal priority, the node must generate one and attach it to the message before multicasting it to other nodes in the group, while also changing the sender. Upon receiving a message, a node must deliver it and add it to the delivery queue. Furthermore, the node must also multicast the message to all other nodes in the group to ensure that all group members receive the message. This custom ISIS algorithm incorporating R-multicast functionality is essential to ensure effective communication within the group and must be developed carefully to ensure its proper functioning.

2. Design ensures total ordering

The system implements the decentralized ISIS algorithm and a multicast approach to ensure that there is a total ordering of messages. Rather than sending replies only to the sender, the system multicasts the proposal priority, which contains the node_name and time of the sender, to all nodes in the group. By doing this, all nodes have the same information about all messages and can update their priority queue accordingly, ensuring that messages are processed in the correct order. To guarantee total ordering even further, the system employs R-multicast, which guarantees that all alive nodes have the same message priority. This approach ensures that messages are reliably delivered, even if there are node failures, as the algorithm takes into account all alive nodes. As a result, the system can handle message delivery in a highly efficient and reliable manner, ensuring that all messages are processed in the correct order, even in challenging network environments.

3. Design ensures reliable delivery under failures

Our system has been designed to ensure that messages are reliably delivered, even in the event of node failures. We have incorporated R-multicast into our system, which guarantees that all nodes will eventually have the same message priority, regardless of any failures. This is achieved by ensuring that a message can only exist in two states: either it is known by all alive nodes, or it is not known by all alive nodes. As a result, any failures are handled effectively. Furthermore, our documentation indicates that TCP errors are utilized to detect any node failures. When a failure is detected, the affected node can remove its priority queue after receiving replies from all other alive nodes. We do assume that nodes will not fail during multicast, which means that either all the correct nodes will know the message or none of them will. This method ensures that messages are reliably delivered even under failures, which is critical in our system's operation. In addition to this, we have implemented other measures to enhance reliability, including regular backups of data, and redundant systems to handle any failures. With these measures in place, we are confident that our system is robust and capable of meeting the needs of our users.

4. How to run our codebase and plot our graph

Open a terminal and input the following instructs (one at a time) to run the four tests in requirements:

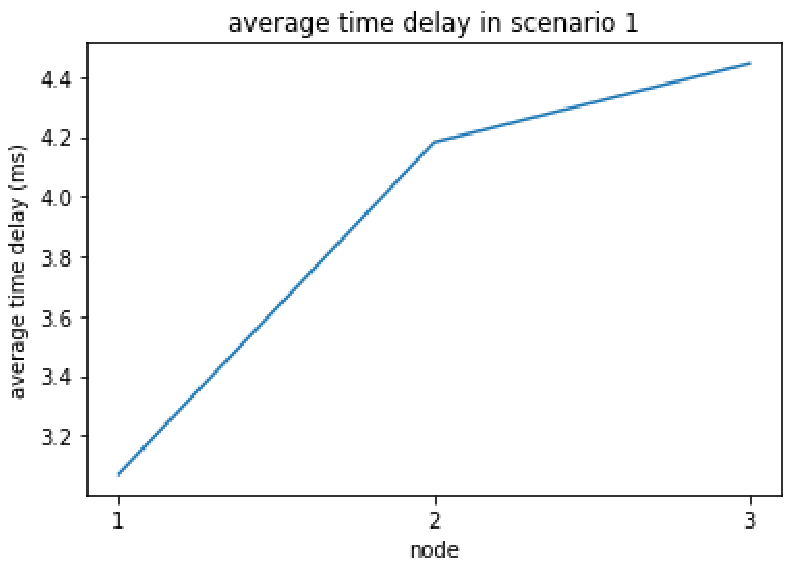
```
sh run3.sh
sh run8.sh
sh run3_kill.sh
sh run8_kill.sh
```

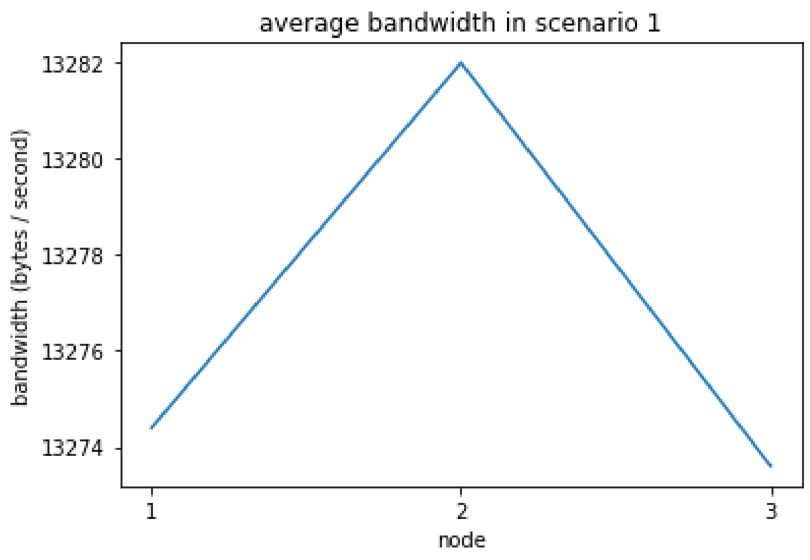
To plot the results, just run the plot.py in the terminal:

```
Python plot.py
```

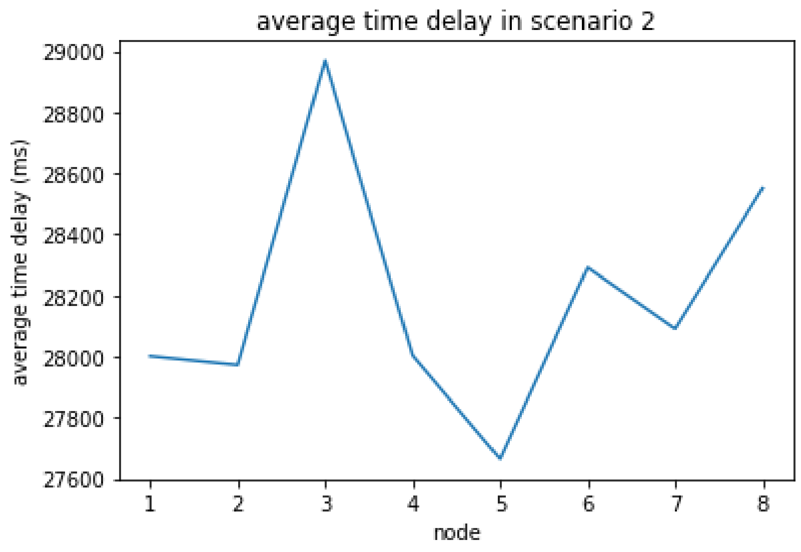
5. Our plotted graphs

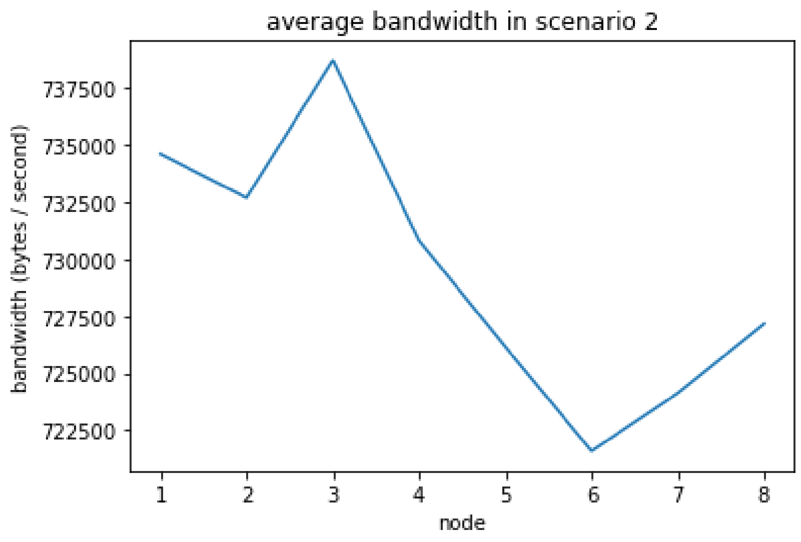
Scenario1:



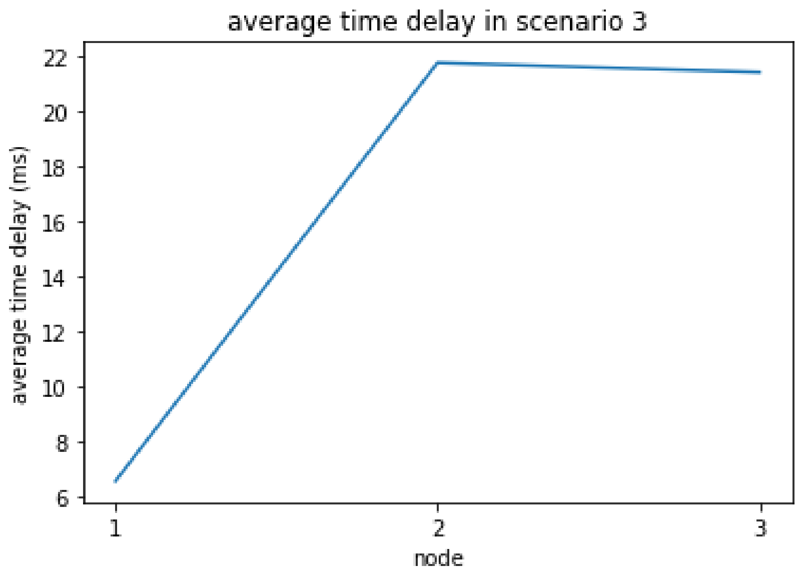


Scenario2:

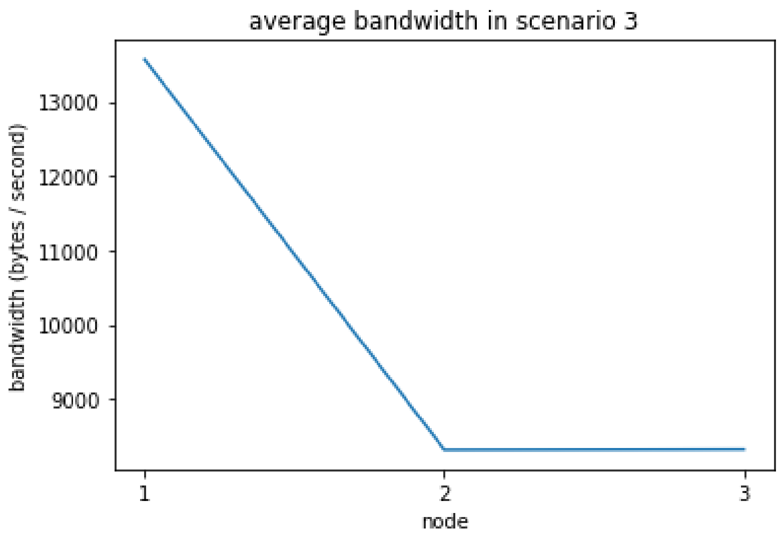




Scenario3:

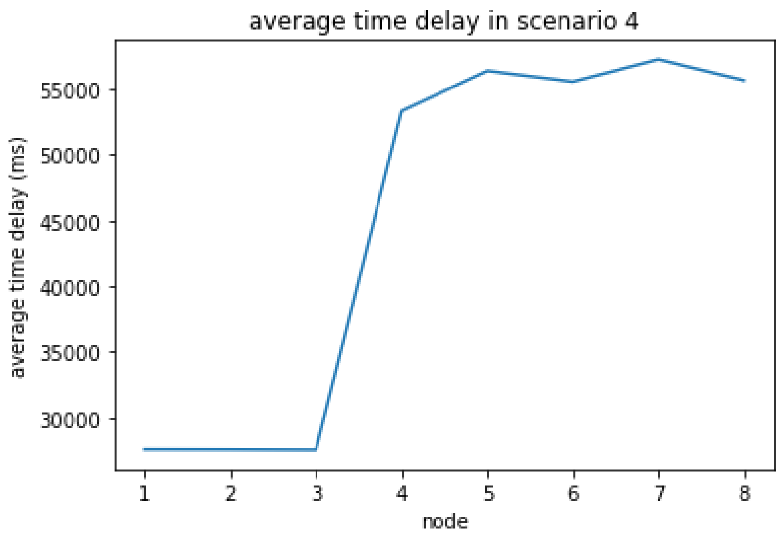


average time delay in scenario 3
average time delay (ms)
22
20
18
16
14
12
10
8
6
1
2
3
node

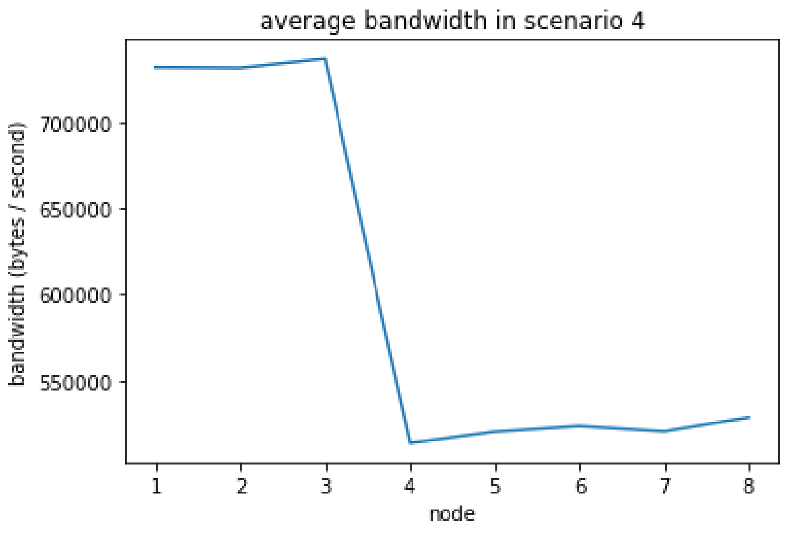


average bandwidth in scenario 3
bandwidth (bytes / second)
13000
12000
11000
10000
9000
1
2
3
node

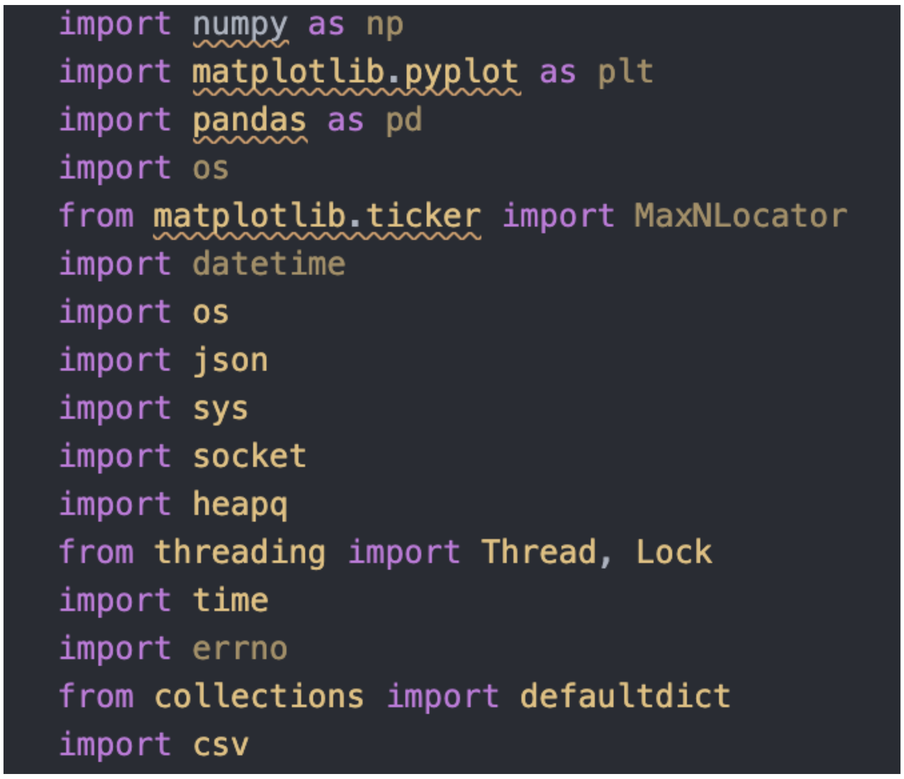
Scenario4:



average time delay in scenario 4
average time delay (ms)
55000
50000
45000
40000
35000
30000
1
2
3
4
5
6
7
8
node



Any packages or libraries that are required to run your code:



```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
from matplotlib.ticker import MaxNLocator
import datetime
import os
import json
import sys
import socket
import heapq
from threading import Thread, Lock
import time
import errno
from collections import defaultdict
import csv
```