MP1: Event Ordering Report

Zhicong Fan(zhicong2), Kehang Chang(kehang2)
03/08/2020
cluster number: g35
Git Revision Number: 8a8b3710f60b377a822776c955275f886cbb6a9e
Running instruction: `go build mp1_node.go && python3 -u gentx.py 0.5| ./mp1_node [number of nodes running] [port number]`

Introduction:

In mp1, we built up a transaction system with up to 9 ATMs working in the same time. In the logger, events, time delay and bandwidth are recorded in order so as to plot the graphs representing the relationship among these clients. In each terminal, the balance of all users would be printed out.

Code Implementation:
ATMs:
Each ATM file(mp1_node.go) consists of 3 parts:

1. When events happen in its own ATM, it multicast to all other processes (Acting like a server, main function)
2. Handle connection among all other ATMs (Acting like a client)
3. Error Detection Subroutine

Main Function:
First of all we run a subroutine for detecting signal interrupts(signal.Notify). Then we set up the connection between this ATM and each other ATM. To achieve this functionality, the ATMs with lower IDs would listen to the ATMs that have higher IDs, and the higher ID ATMs would dial to the lower ID ATMs. Once all the connections are set up, we check our interrupted flag to make sure the current ATM is not terminated by ctrl + c. Otherwise it would not multicast the events happen in its ATM. If this ATM is alive, each event generated would be pushed into the priority queue(pq) and also reordered in the queue. At last, the event would be multicast asking for propose message from other ATMs.

Handle Connection:
In my structure, there are 4 types messages that an ATM would receive:

1. A **REQUEST** message: for which it should push the received event into its own priority queue and send propose back to sender
2. A **PROPOSE** message: when an ATM receive a **PROPOSE** message, it would than compare the received priority with its own priority and if the received one is larger, update that in the priority queue. Moreover, it would go through a Propose map to see if

this message has received all the proposes from other ATMs. Suppose we now have received all the propose messages and the event is at the head of the priority queue, we would set this message to deliverable and update the balance array. At last we multicast **AGREE** message and pop the consecutive deliverable events from the queue.

3. An **AGREE** message: when an ATM receive an **AGREE** message, it would then update the priority of the event in the priority queue and also pop all the consecutive deliverable events if the first event in the queue is deliverable.
4. A **BROKEN** message: when an ATM receive a **BROKEN** message, it would then set the corresponding ATM to false in the "isAlive" array and won't wait for the propose from this ATM anymore. Besides this, it would also pop out all the events that were sent from the broken ATM so that the priority queue would not be stuck.

Error Detection Subroutine:

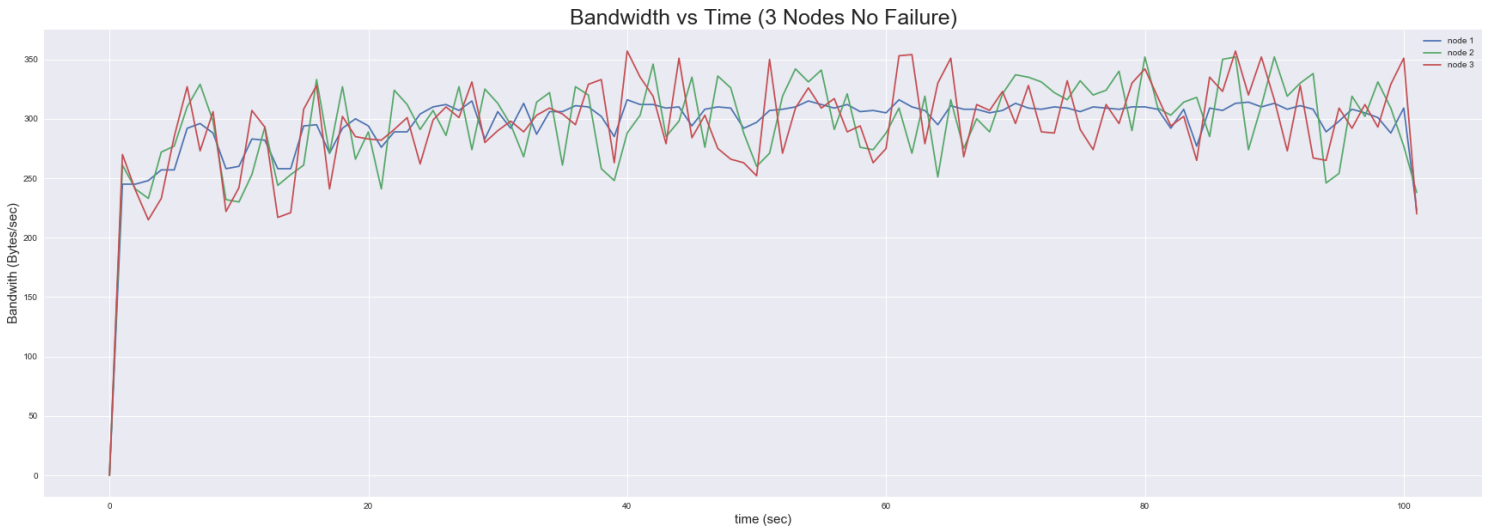
An error Detection Subroutine would be initialized at the start of the program, and once an interrupt signal is detected, it would multicast to all other ATM a **BROKEN** message which contain the ID of its ATM and shut down all the current running routines, so that no more events generated in this ATM would be multicast.

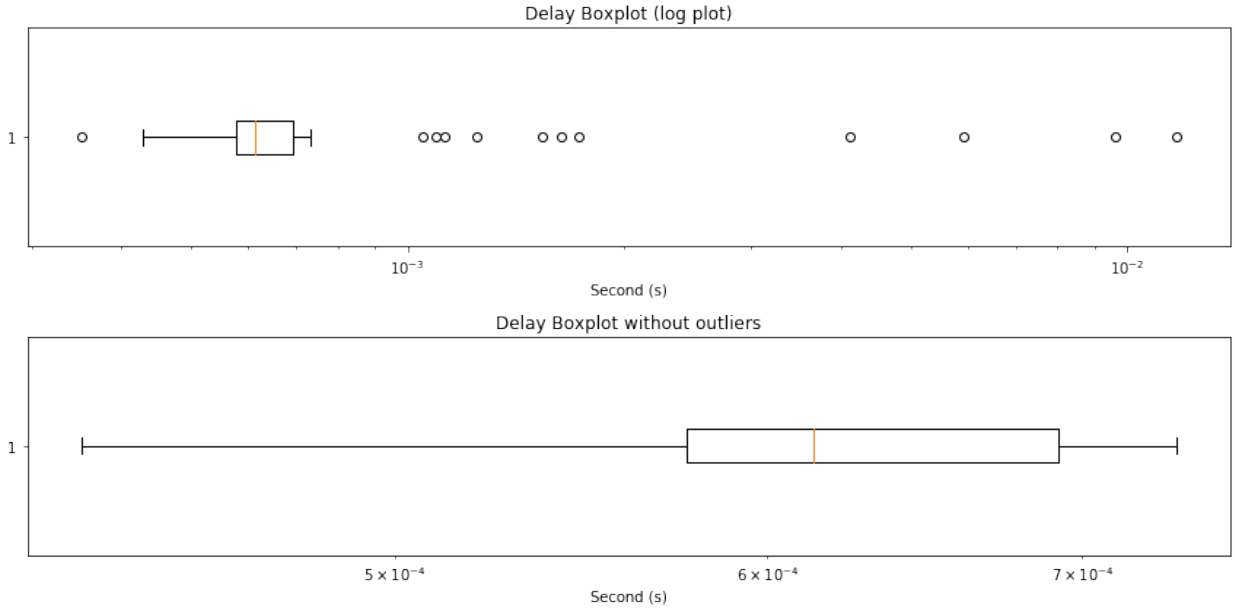
Design Document:

We used ISIS algorithm to ensure total ordering of the event. We encode the Agreed, Propose, Request packages as the same format below: [msg type] [event id] [proposed priority][senderid][msg]. We send all locally generated events in main() function and handle different types of packages in gohandleconnection() subroutine once a secure TCP channel has been established. After one locally generated event is being sent, it will receive Propose packages from all other processes, and then it will send out Agreed packages to all other processes from original sender. Thus the priority queue's total ordering is ensured.

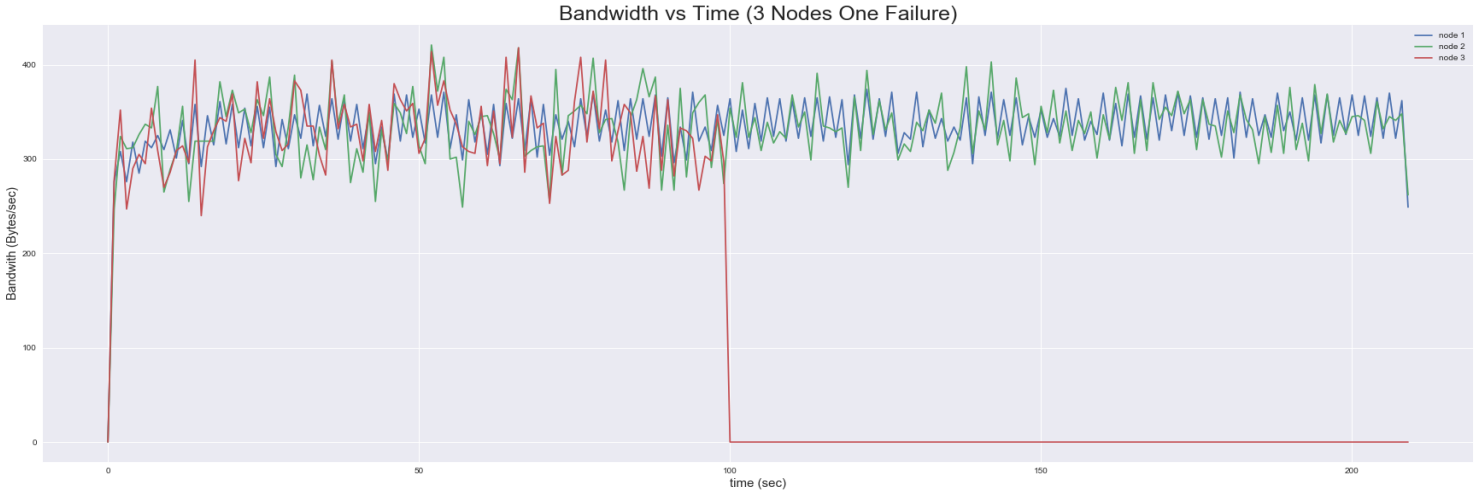
We used TCP protocol to ensure the secure network connection among all processes. We also implemented a (Notifying) func subroutine to detect any keyboard interrupt in real time. If a process is being interrupted by keyboard, it will send out "error msg" before it cuts down network connection with other node, thus one node failure won't bring the whole system down. Also we used a global variable *interrupted* to ensure packages handling routine won't be interrupted in the middle by keyboard.

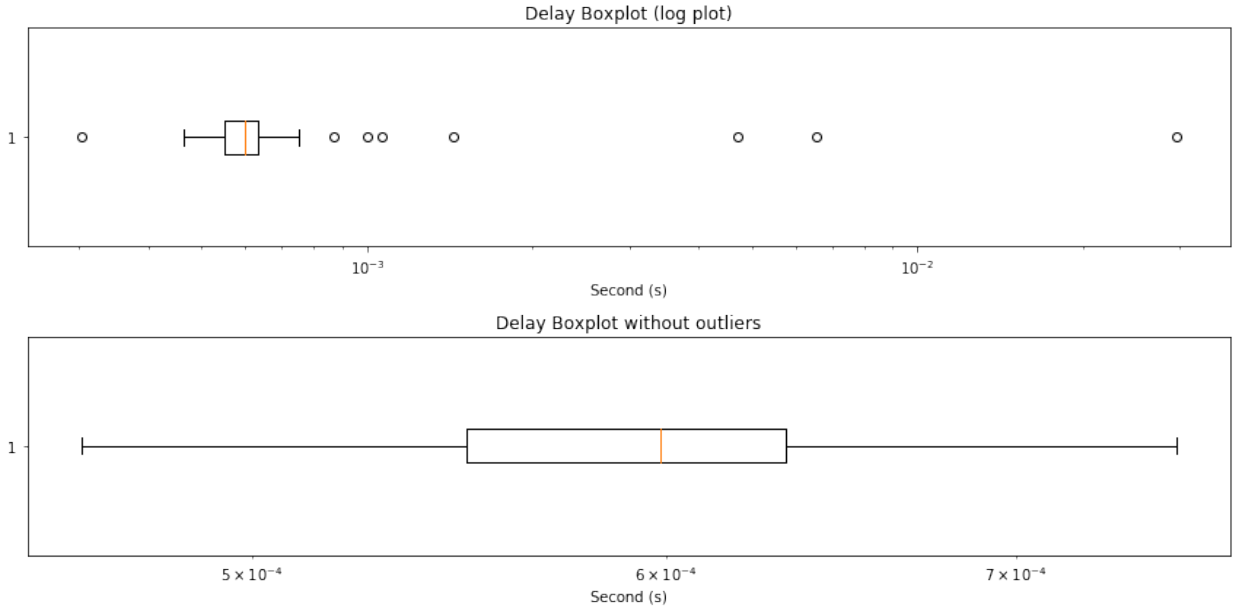
Graph:


Bandwidth vs Time (3 Nodes No Failure)
Bandwith (Bytes/sec)
time (sec)
node 1
node 2
node 3
0
50
100
150
200
250
300
350
0
20
40
60
80
100

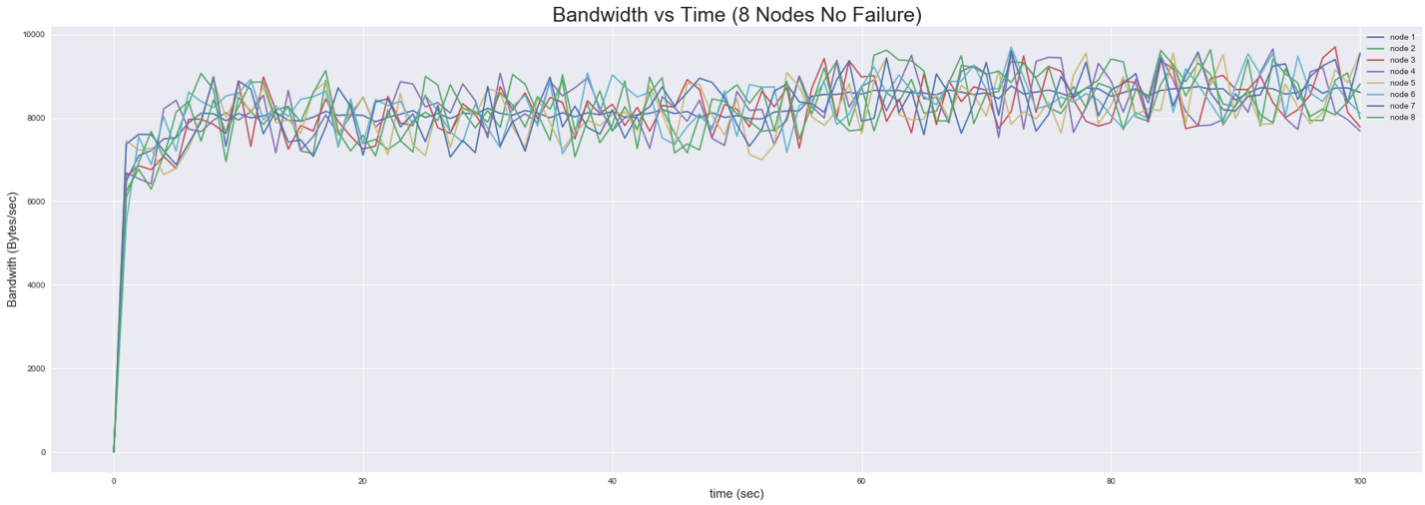


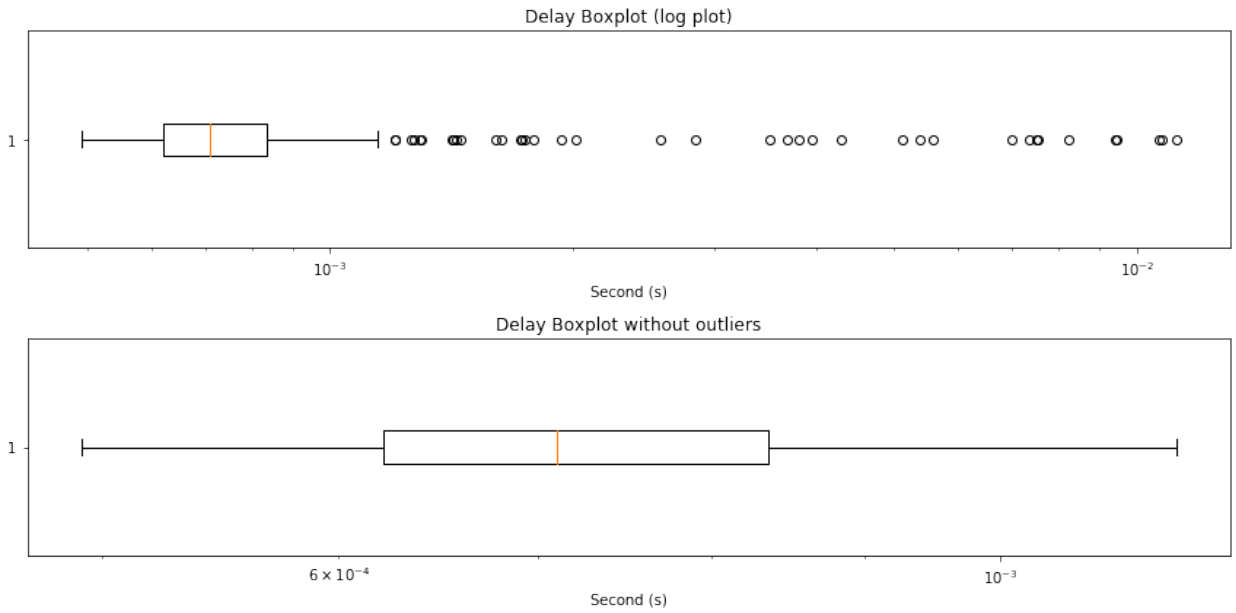
(3 Nodes Delay No Failure)



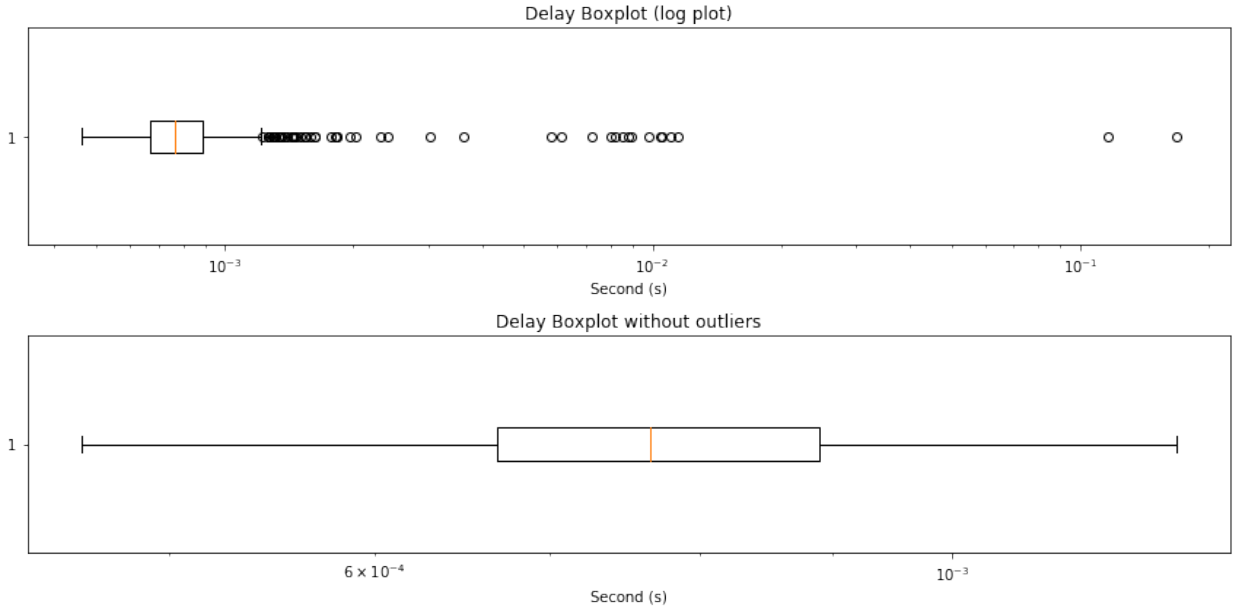


(3 Nodes Delay One Failure)





(8 Nodes Delay No Failure)



(8 Nodes Delay Three Failure)

Language used: “Go” for ATMs, “Python” for plotting the graph