**REPORT**

**Assumptions:** There are no failures of any processes.

**Message Format:**

msg = {

"data": 'a',

"conn\_type": "tom",#peer2peer or total order multicast

"type": "app",#app or data or ack

"timestamp": 0,

"ack\_flag": 0,#check for ack

"msg\_id": random.randint(1, 10000),#randon integer

"ports": [6002] #destination

}

**Design Details and Pseudocode:**

I have an application layer and a middleware layer.

In the code, when I call init\_cluster(). It spawns n number of processes passed as an argument. Once each process is spawned, it starts its middleware in a thread and on other thread, it keeps listening for an output from the middleware.

Middleware – Server class, it has a constructor which initializes a socket connection.

Anytime if I send a message from my tester function, it receives a message and then looks for the type of message it is.

Peer-to-Peer:

If it is a peer to peer message, then a destination port is found from the message, and a socket connection is used to send the message.

After the message is received on the destination server, it displays the message with its self- port to determine who received the message.

Total Order Multicast:

Once a message is received, there are three types – “app” – message from application layer, “data” – multicasted message and “ack” – acknowledgement message.

If it is app, we first multicast this message to all processes including self, increment the timestamp and send message with this timestamp.

If it is data, we put this message in our queue and then sort this queue,

For the head of queue, multicast acknowledgement and mark that acknowledgement for the head of queue is sent.

If it a ack, we check if it is in our ack dictionary which keeps the count of our dictionary. If it is, we simply increment the count or else we add this new key in the dictionary.

A while loop keeps track if the head of queue has received all the acks and then pops it from queue and sorts again, if not, multicasts ack for head.

In every dictionary we use msg\_id as the key for unique message tracking.

Test Cases:

1. The test cases involved, sending just one message in total order multicast and peer to peer.
2. Sending two messages simultaneously to 2 processes, which maintain the order of the message in total order multicast and in peer to peer displays the message.
3. The final output is printed on the screen as a list element.