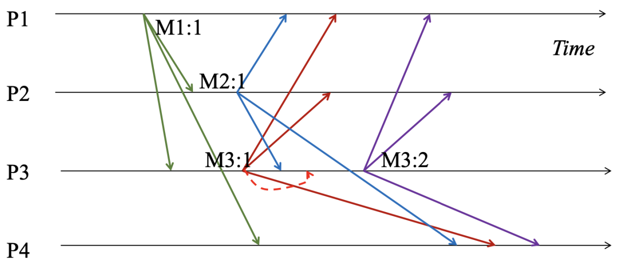
COL724 - Assignment 2

Totally Order Multicast

Gurarmaan S. Panjeta
2020CS50426



Algorithm & Report & Testing Results

October 13, 2023

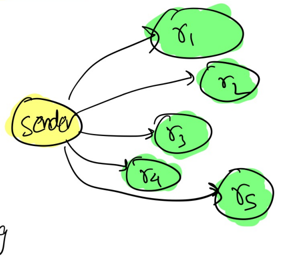
The Total Ordering Algorithm & Pseudo-Code

Total Order Multicast Algorithm

Two-phase Multicast

Dissemination Phase

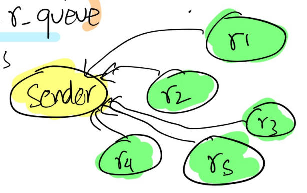
Step1. Sender Sends message to recvs.



Step2. Recvs. append message to respective reception queues after timestamping

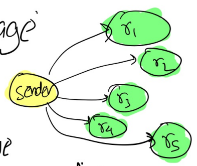
```
For recv in recvs:
    increment recv. local_timestamp
    append message to recv. r_queue
```

Step3. Recvs send local timestamps back to the sender



Decision Phase

Step4. Sender Sends 'validation-message' containing max. of the recieved time stamps



Step5. All recievers ubdate their r_queue with timestamp value of message recieved in 4. message is marked "deliverable"

Step6. All recievers reorder their r_queue in ascending orders of time stamps

Step7. All messages at the head of the queue That are deliverable are now delivered to the upper layer.

The Two-Phase Multicast algorithm that delivers messages totally-ordered.

Implementation Details

Class abcast_node

Built to simulate a communicating node. Has the following attributes :-

- local_timestamp - Indicates serial number used for indexing received messages
- r_queue - To store a priority Queue of messages, and dequeue them when deliverable.
- delivered_messages - Represents the order of messages passed to the upper layers
- id - Unique Number per node
- stored_ts - Contains information to avoid race conditions

An abcast_node can perform the following methods to manipulate it's state :-

- reorder_r_queue(), when an update may potentially require reodering
- update_r_queue(message, valid_ts), when a message receieves it's commit time, update local time stamp and mark it deliverable
- clean_queue() - remove and deliver messages at the head of the queue if deliverable
- receive_message(message) - enqueue in r_queue and update local timestamp
- receive_uni(message) - for handling unicast messages
- return_local_timestamp_message() - for reporting local timestamp to sender

Class abcast_system

Built to simulate a system of communicating nodes. Has the following attributes :-

- nodes - Instances of the abcast_node class
- n_nodes - Number of nodes in the system
- global_clock - To study order of events
- event - To handle race conditions

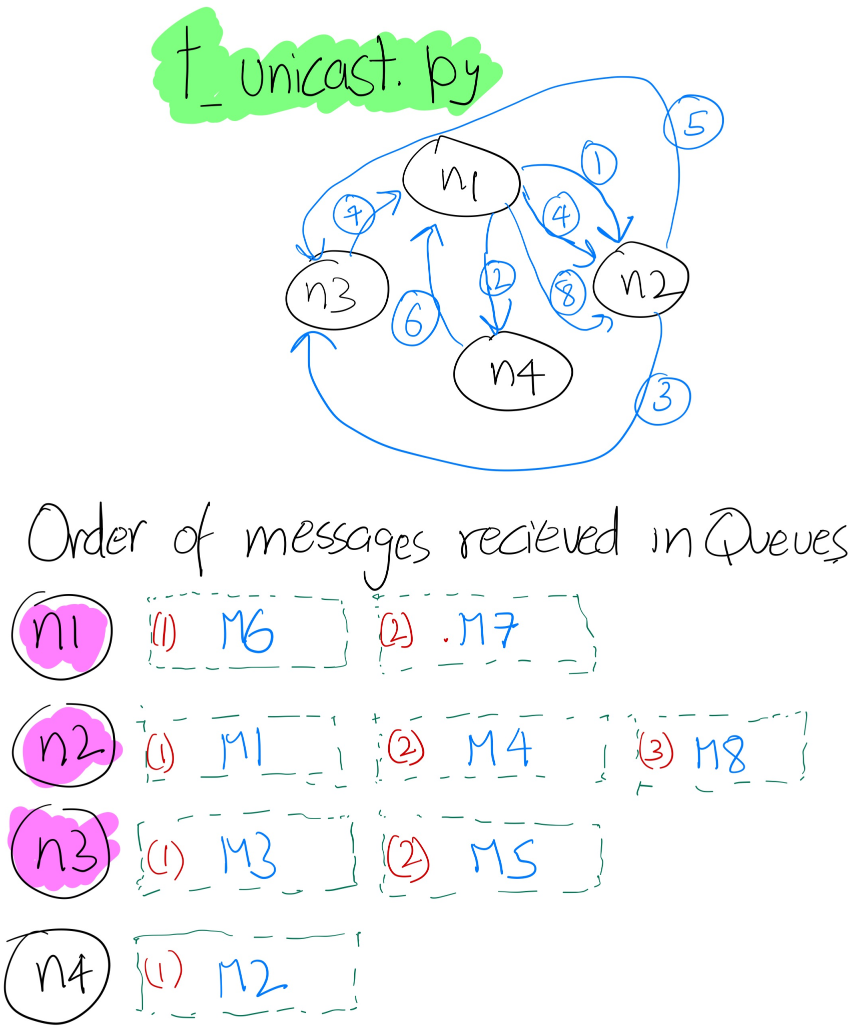
An abcast_system can perform the following methods to communicate within nodes :-

- start_global_clock() - A daemon that runs in the background and maintains the global clock
- transmit(message, n_from, n_to, rec_delay) - Send Unicast from n_from to n_to
- emit(message, n_from, ns_to, rec_delays, reply_delays, commit_delays) :- Send multicast from n_from to all nodes in n_to, and with delays passed as arguments
- schedule_uni(node_num, induced_delay, message) and schedule_action(node_num, action, induced_delay, message, validation_ts) :- To transmit messages at the delays required.

Testing

Unicast Testing

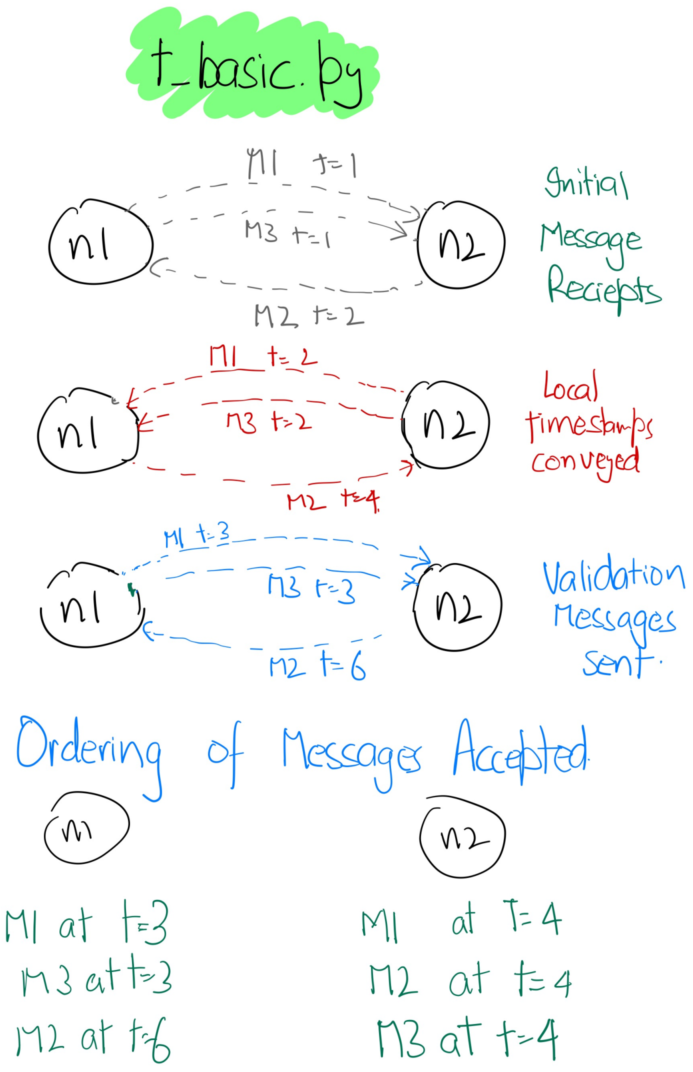
Objectives and Observations - Demonstrates unicast-ability between nodes, and displays that the order of receipt of messages sent via unicast is as per the delays encountered within the links they were sent through.



Reliable Unicast Testing and Results

Multicast Hello World!

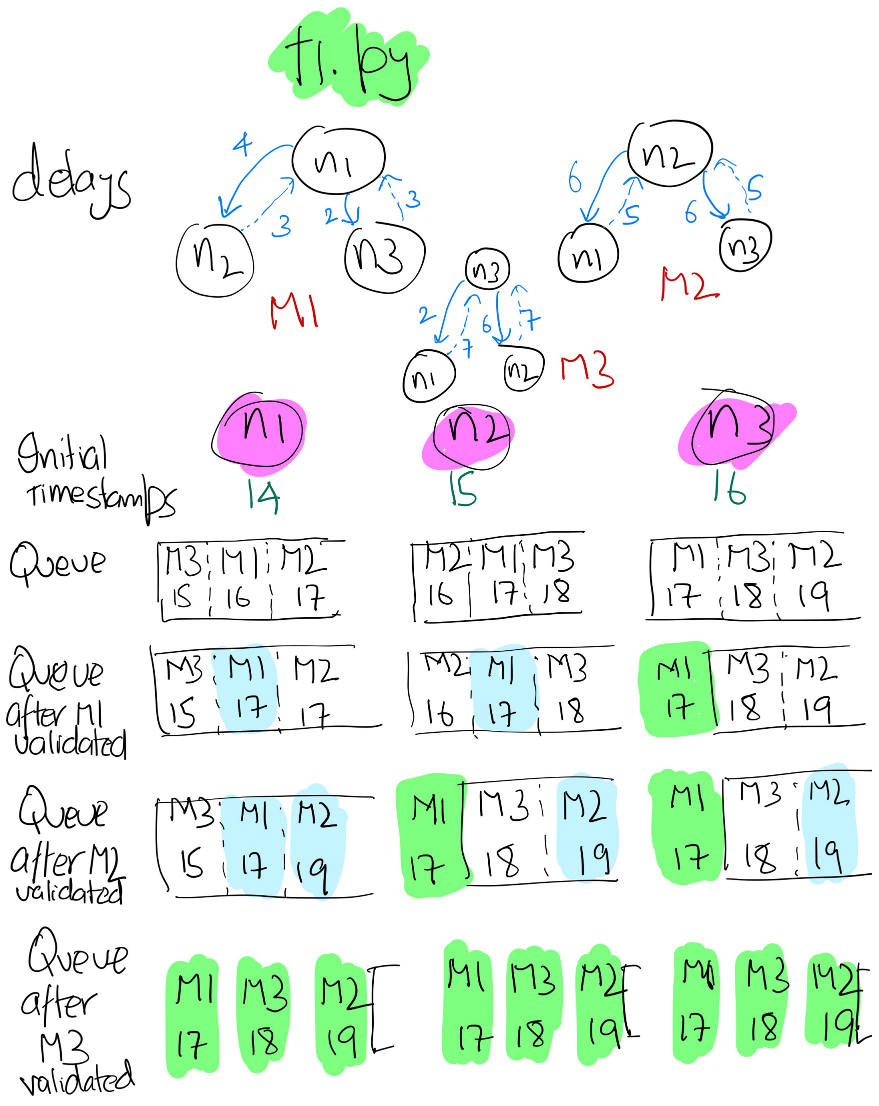
Objectives and Observations - simulates two nodes, with variable delays between messages sent in either direction. The order of messages passed to the application layer is the same in both the nodes, as are their *committed* timestamps, thus showing atomicity and total ordering.



2 Nodes Multicasting and preserving Total Order

Multicast - Multi Nodes and Multi Messages

Objectives and Observations - 3 Nodes with only Multicast Traffic. All communication of different messages overlaps with each other yet the mechanism is able to determine required order! Final Print statements show the order of messages in which they are passed to the delivery queue. Interim messages printed show the timestamp-communication mechanism at work.

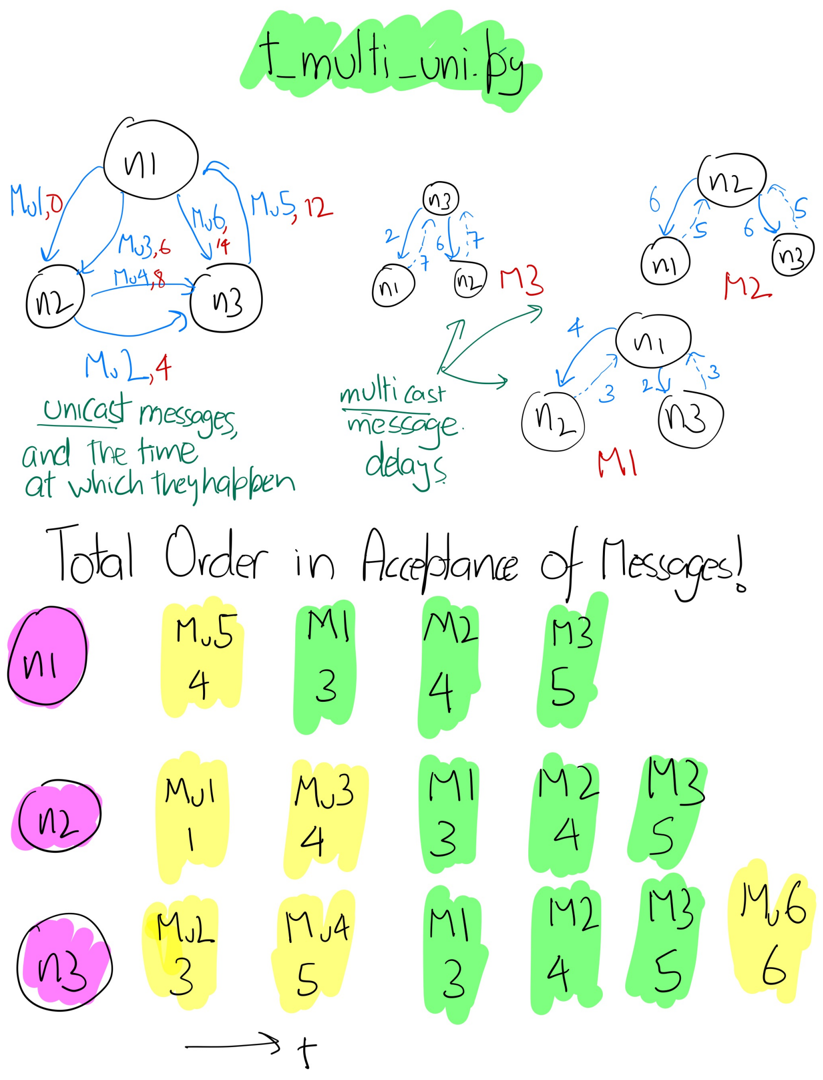


3 Nodes Multicasting, overlapped communication, yet Totally Ordered

Note :- The total order of messages is different from real-time order, as can be observed from the real time reciept of messages and the final order in which they are passed to upper layers.

Multicast - Overlapped with Unicast

Objectives and Observations - 3 Nodes with Multicast Traffic, with unicast messages interspersed in between. We can observe that the multicast messages retain ordering despite unicast messages fiddling with timestamps! All communication of different messages overlaps with each other yet the mechanism is able to determine required order. Final Print statements show the order of messages in which they are passed to the delivery queue.


t_multi_uni.py
n1
n2
n3
Mu1,0
Mu3,6
Mu4,8
Mu6,14
Mu5,12
Mu2,4
unicast messages, and the time at which they happen
multicast message delays
M3
M2
M1
Total Order in Acceptance of Messages!
n1: Mu5 4, M1 3, M2 4, M3 5
n2: Mu1 1, Mu3 4, M1 3, M2 4, M3 5
n3: Mu2 3, Mu4 5, M1 3, M2 4, M3 5, Mu6 6
t

3 Nodes Multicasting and Unicasting, yet Totally Ordered!