CS425 MP2 Report:

# Overview:

In this project, we implement the ordered multicast mechanism through building the protocol layer by layer. Basically, we build the reliable unicast protocol by adding timer, ack message, and retransmission on top UDP. Then, the basic multicast feature is built based on reliable unicast protocol. Finally, the total ordered multicast and causal ordered multicast protocols are implemented using basic multicast protocol.

# Reliable unicast protocol

This protocol is implemented through UDP. Since UDP do not guarantee the destination will receive the message sent, we require each peer to send back an ack message to sender whenever it receives an incoming message. Since both normal message and ack message can be loss, timer is required for each sent message on sender.

Whenever a message is going to be sent, sender will setup a timer that will expire in around 10 seconds. If a ack message of the normal message is received, the corresponding timer will be deleted; otherwise the previous message will be re-sent to destination and timer will be re-scheduled.

# Basic unicast protocol

This protocol is built on top of reliable unicast protocol. For sender, it will selected out the peers belong to the target group and use reliable unicast to transfer messages to all group members. For receiver, whenever it receives a group message, it will check whether it belong to target group, if so it will deliver this message to high layer, otherwise the message will be discarded.

# Causal order multicast

This feature is built based on basic multicast protocol. It includes a time vector locally and to each message sent.

Initially, the values inside time vector V are 0s. Whenever a peer *i* want to send a message, it will increase the value of V[i] by 1, piggyback the time vector V into the message, and multicast the message to all group members through basic multicast.

When the peer i receive a message <Vj, m> from peer j, peer i will place <Vj, m> into holdback queue and delivery this message until following condition satisfied:

,

then message <Vj, m> can be delivered and should be increased by 1.

Since the time vector captures the causality locally and inside the message, this protocol can naturally guarantee the messages delivered are in causal order.

# ISIS total order multicast

ISIS total order multicast protocol is built on top of basic multicast protocol.

The multicast sender multicasts message to all members inside the target group.

Recipients add the received message to a special queue called the priority queue, tag the message undeliverable, and reply to the sender with a proposed priority (i.e., proposed sequence number). Further, this proposed priority is 1 more than the latest sequence number heard so far at the recipient, suffixed with the recipient's process ID. The priority queue is always sorted by priority.

The sender collects all responses from the recipients, calculates their maximum, and re-multicasts original message with this as the final priority for the message.

On receipt of this information, recipients mark the message as deliverable, reorder the priority queue, and deliver the set of lowest priority messages that are marked as deliverable. In order to break tie, if there are two messages with same priority, the one that is undeliverable will be put on the head. If two messages have same priority and both deliverable, then put the one with low sender ID on the head. If two messages have same priority, sender ID and both deliverable, then put the one with low message ID on the head.

# Total Order Multicast with Sequencer

Total Order Multicast with Sequencer is also built on top of basic multicast.

The Chat clients multicast their message to all Chat clients but also a separate sequencer process. The sequencer listens for Chat messages and upon receipt assigns each one a total order sequence number and multicasts an Order message to Chat clients with the same message but new order number. The sequencer starts with a sequence number of 0 and increments each time it multicasts an Order message.

The Chat clients queue all Chat and Order messages and update corresponding Chat messages with their total order sequence number in the holdback queue. The Clients also have a thread running that periodically checks the holdback queue for the message with the next total order sequence number, starting from 0. Once next message is found, it gets printed, removed from the holdback queue, and the thread will keep checking for the message with the next sequence number.