# ECE 428 MP1 Design Documentation

Josiah McClurg and Yihua Lou

***Abstract*—The reliable multicast protocol detailed by this document is a demonstration of several key distributed systems concepts. In particular, the concepts of ordering guarantees and failure detection are put into practice, along with the establishment of several liveness and safety properties of the system. The purpose of this document is twofold. Primarily, it contains a concise description of the reliable multicast (rmcast) protocol we have developed and serves to place the protocol on a strong theoretical basis by sketching proofs of the Causal Ordering, Reliable Multicast, and Failure Detection system properties the protocol seeks to guarantee. Secondly, it discusses details and limitations of the C++ protocol implementation submitted with the assignment.**

## I. Protocol

The protocol defines the following objects:

- $p_i$ Identifier for process $i$.
- $G$ Identifier list $G$ such that $p_i \in G \iff$ process $i \in$ multicast group.
- $T_G$ Vector timestamp containing pairs $t_i \forall i = p_i \in G$, where $t_i$ is the timestamp of process $i$.
- $D_{G,k}$ Vector acknowledgment list containing elements $a_i = n \forall i = p_i \in G$ where $n$ is such that $m_{n,i\to k}$ is the last message from process $i$ that process $k$ has delivered.
- $m_{n,i\to j}$ Application message $n$ from process $i$, destined for process $j$.
- $\heartsuit_{i\to j}$ Rmcast heartbeat message.
- $\diamondsuit_{\{n,k\},i\to j}$ Rmcast retransmission request from process $i$ asking process $j$ for message $m_{n,k\to i}$.
- $m'_{n,i\to j}$ Rmcast message containing $m_{n,i\to j}$, along with the timestamp $T_G$ and acknowledgment list $D_{G,k}$ of process $i$ at transmission time.
- $M_j$ Message list $M_j$ such that $m'_{n,i\to j} \in M_j \forall i = p_i \in G \iff$ message $m_{n,i\to j}$ has been received and/or delivered by process $j$.

**Algorithm 1** Reliable multicast send.

**procedure** MULTICASTSEND($m_{n,self\to j} \forall j \in G$)
  increment($T_G$)
  deliver($m_{n,self\to self}$) ▷ Optimistically self-deliver.
  update($D_{G,self}$)
  $m'_{n,self\to self} \leftarrow m_{n,self\to self}, T_G, D_{G,self}$
  $M_{self} \leftarrow m'_{n,self\to self}$
  **for** $p_i \neq p_{self} \in G$ **do**
    $m'_{n,self\to i} \leftarrow m_{n,self\to j}, T_G, D_{G,self}$ ▷ Encode multicast headers.
    unicast($m'_{n,self\to j}$)
  **end for**
  $n = n + 1$ ▷ Increment sequence number.
**end procedure**

**Algorithm 2** Failure detect and heartbeat.

**procedure** FAILUREDETECT(resettable timer $T_i$ and heartbeat period $H_i$ for each $p_i \in G$.)
  **repeat**
    **for** $p_i \neq p_{self} \in G$ **do**
      **if** $T_i$ has expired **then** ▷ Declare process as failed.
        removeFromGroup($p$,$G$)
      **else if** $H_i$ has elapsed **then** ▷ Send heartbeat if needed.
        $\heartsuit_{self\to i} \leftarrow D_{G,self}$
        unicast($\heartsuit_{self\to i}$)
      **end if**
    **end for**
  **until** end of program.
**end procedure**

### A. Proof of Causal Ordering

Prove: If multicastSend($m$) happens-before multicastSend($m'$), and $m'$ delivered by correct process $p$, then for process $p$, deliver($m$) happens-before deliver($m'$).

For vector timestamps $T_1$ and $T_2$, it can be proved that $T_1 < T_2 \Rightarrow T_1$ happens-before $T_2$. This fact, and the fact that the delivery ordering of mutually-concurrent events is irrelevant is used to create the causalitySort function referred to on line 28 of Algorithm 3.

The behavior of line 38 guarantees that deliverable messages which happened-after currently-undeliverable messages are not delivered.

### B. Proof of Reliable Multicast

Prove the Integrity, Validity and Agreement properties of the reliable multicast algorithms.

*1) Integrity:* Prove: (a) Each message delivered at most once. (b) The sending process is a member of the message's multicast group, and (c) the message was sent by its claimed sender.

(a) is proved by contradiction. Given that a message has been delivered once, assume that the same message is delivered a second time. Line 27 of Algorithm 3 implies that the acknowledgment list $D_{G,self}$ was not updated after the initial delivery, which is contradicted by line 36 of the same algorithm.

(b) and (c) are ensured by assuming non-corrupting channel and unique process IDs. Each message is tagged with sender information, and the protocol ignores incoming messages whose sender is not in $G$. See line 12 of Algorithm 3.

*2) Validity:* Prove: Eventual delivery of all sent messages to own process.

See line 3 of Algorithm 1.

*3) Agreement:* Prove: If a message is delivered to one process, it is delivered to all.

If a message is delivered to some process $p_i$, line 4 of Algorithm 1 and line 36 of Algorithm 3 guarantees that $D_{G,i}$ contains this information. Line 7 of Algorithm 2 indicates that this delivery information is eventually transmitted to each other process in $G$. If the information is not transmitted, the failure detection ensures removal of $p_i$ from $G$.

Now, all correct processes are guaranteed to have information about the deliveries of other processes within some finite time. Assuming periodic receipt of new chat messages, and reasonable reliability of channel, the retransmission behavior of lines 4 and 47 of Algorithm 3 indicates a high probability that retransmission requests will eventually be made and serviced. This can be made into a hard guarantee through the use of additional timeouts, or by piggybacking retransmission requests onto heartbeat messages.

### C. Proof of Failure Detection

Prove: Every failure is eventually detected.

Given that process $p$ has failed, it will not send out heartbeats. Algorithm 2 guarantees that each process will detect this within a finite time. Because delays can be unbounded, there is no guarantee against false positives in the failure detection.

## II. Implementation

As is suggested by line 17 of Algorithm 3, the C++ implementation does not currently support additions to group membership. However, the manner in which failures are communicated (namely, by transmitting a failed nodes list in the message header) allows this restriction to be relaxed in future revisions.

Rather than storing all messages in a single message store $M_{self}$, the C++ implementation maintains its own message list and each list of messages received from other nodes separately. $D_{G,self}$ is not implemented as a single object, but the information contained is distributed across the same distinct global state structures that contain the stored messages. Each $D_{G*,j}, j \in G$, however is implemented as a single structure.

Also, different from what is shown here, the same header information is placed into heartbeats and receive request as is placed into message headers, and timeout is reset with the receipt of any message. This aids the speed of communication and helps prevent unnecessary timeouts. The message headers are encoded to and decoded from a simple binary format, which conserves channel bandwidth.

**Algorithm 3** Reliable multicast receive

1: $\triangleright$ Retransmission request.
2: **procedure** MULTICASTRCV($\diamondsuit_{\{n,k\},i\to self}$)
3: **if** $m_{n,k\to i} \in M_{self}$ **then**
4: unicast($m_{n,k\to i}$)
5: **end if**
6: **end procedure**
7:
8: **procedure** MULTICASTRCV($\heartsuit_{i\to self}$) $\triangleright$ Heartbeat.
9: resetTimeout($p_i$)
10: **end procedure**
11: $\triangleright$ Application message.
12: **procedure** MULTICASTRCV($m'_{n,i\to self}$ s.t. $p_i \in G$)
13: increment($T_G$)
14: $m_{n,i\to self}, T_{G'}, D_{G',i} \leftarrow m'_{n,i\to self}$
15: merge($T_G$,$T_{G'}$)
16: $\triangleright$ Fail if someone else thinks you are failed, or if you have already delivered messages from the node they think has failed.
17: **for** $p_j \in G$ s.t. $p_j \notin G'$ **do**
18: markSuspectedFailure($p_j$)
19: **if** $p_j = p_{self}$ or
20: $a_{j,self} \in D_{G,self} > a_{j,i} \in D_{G',i}$ **then**
21: quit();
22: **end if**
23: **end for**
24: $\triangleright$ Store message. Disallow duplicates.
25: $M_{self} \leftarrow m'_{n,i\to self}$
26: $\triangleright$ Find deliverable messages.
27: $Q \leftarrow m'_{k,j\to self} \in M_{self}$ s.t. $k = a_j{+}1, a_j \in D_{G,self}$
28: causalitySort($M_{self}$) $\triangleright$ Sort according to causality.
29: **for** $m_x \in M_{self}$ **do**
30: **if** $m_x \in Q$ **then**
31: $m_{k,j\to self} \leftarrow m_x$
32: $\triangleright$ Only deliver messages from suspected failure nodes if doing so is needed to ensure agreement.
33: **if** notIsSuspectedFailure($p_k$)
34: or $\exists a_j \in \cup_{t\in G} D_{G^*,t}$ **then**
35: deliver($m_{k,j\to self}$);
36: update($D_{G,self}$)
37: **end if**
38: **else if** $m_x$ happens-after $m_{x-1}$ **then**
39: break
40: **end if**
41: **end for**
42: $\triangleright$ Send retransmission requests to everyone who has delivered the messages we are waiting for.
43: **for** $m_{q,j\to self} \notin M_{self}$ s.t.
44: $\exists$ undeliverable $m_{q',j\to self} \in M_{self}$ **do**
45: **for** $a_k \in \cup_{t\in G} D_{G^*,t}$ s.t. process $k$
46: has delivered $m_x$ **do**
47: unicast($\diamondsuit_{\{q,k\},self\to j}$)
48: **end for**
49: **end for**
50: $\triangleright$ Remove from $M_{self}$, the messages which are known to be delivered by everyone. This is determined by considering each $D_{G^*,t} \forall t \in G$.
51: cleanup($M_{self}$)
52: **end procedure**