ECE 428 MP1 Design Documentation

Josiah McClurg and Yihua Lou

February 25, 2013

Abstract

1 Introduction

2 Methods

For by Algorithms 2, 3, and 1, define the following variables: Multicast group G^k, own process identifier p_{self}, raw message m, sequence number s, message source p_s, message with meta information m', along with current vector timestamp T^k, and delivery acknowledgment set $D^{k \times k}$, delivered message store $S^{k \times w}$, holdback queue $Q^{k \times z}$, last delivery vector timestamp T_l^k, and timeout list L^k all indexed by $p \in G$.

2.1 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. By this reasoning, Algorithm 3 ensures that the most recently delivered message was either sent before or was sent concurrently with each new message that is delivered.

2.2 Proof of Reliable Multicast

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

2.2.1 Integrity

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

(a) is easily proved by contradiction. Given that a message has been delivered once, assume that the same message is delivered a second time. Algorithm 2 guarantees that the sequence numbers of both messages are the same. This implies that the acknowledgment list D was not updated after the initial delivery, which is contradicted by Algorithm 3.

(b) is deferred to the underlying process communication protocol.

(c) is ensured by allowing processes other than the original sender to retransmit any delivered message (in case the original sender has failed).

2.2.2 Validity

Prove: Eventual delivery of all sent messages to own process.

Deferred to underlying process communication protocol.

2.2.3 Agreement

Prove: If a message is delivered to one process, it is delivered to all.

If a message is delivered to a process, Algorithm 1 guarantees that all correct processes are aware of this delivery within some finite time. Thus, all correct processes can eventually detect any missing messages.

Now, Algorithm 3 requests retransmission of missing messages until it receives the needed messages. the property is proved, provided that the network does not selectively delay message retransmissions without bound while continuing to speedily deliver heartbeat messages.

2.3 Proof of Failure Detection

Prove: Every failure is eventually detected.

Given that process p has failed, it will not send out heartbeats. Algorithm 1 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.

3 Conclusion

Algorithm 1 Failure detect thread.

```
procedure failureDetectThread(G, p_self, Q, T, T_l, D, L)

    repeat
        for p ≠ p_self ∈ G do
            if time() − L[p] ≥ T_f then                ▷ Declare process as failed.
                removeFromGroup(p,G,Q,T,T_l,D)
            else if time() − L[p_self] ≥ T_h then       ▷ Send heartbeat if needed.
                m′ = piggyback(T, 0, D[p_self], p, ♡)
                unicast(p,m′)
            end if
        end for
        t = T_h − (time() − L[p_self])
        sleep(min(t,0))
    until end of program.
end procedure                                          ▷ Updates G,Q,T,T_l, and D
```

Algorithm 2 Reliable multicast send.

```
procedure multicastSend(G, p_self, m, s, T, D)
    for p ≠ p_self ∈ G do
        incrementTimestamp(p_self,T)
        m′ = piggyback(T, s, D[p_self], p, m)
        unicast(p,m′)
    end for
    incrementSequenceNumber(s)
end procedure                                          ▷ Updates s, and S.
```

Algorithm 3 Reliable multicast receive

procedure MULTICASTRCV(G,D,T, p_{self}, p_s, m', S, Q, T_l, L)
    $\{T_m, s, D[p_s], p_{from}, m\}$ = unpiggyback(m');
    mergeTimestamps(T,T_m) ▷ Ensure consistency of timestamps.
    $L[p_s]$ = time() ▷ Reset timeout counter for p_s.
    ▷ Ensure consistency of group membership.
    for $p \in D[p_s]$ s.t. $p \notin D$ **do**
        removeFromGroup(p,G,Q,T,T_l,D)
    end for
    ▷ Delete messages known to be delivered to everyone.
    for $l \in S[p] \forall p \in G$ s.t. $l.s \leq \min(D[p])$ **do**
        removeFromMsgStore(l,S)
    end for
    if $m == \Xi$ **then** ▷ This is a retransmission request message.
        discard(m)
        for $l \in S[p_{from}]$ s.t. $l.s \leq s$ **do**
            m' = piggyback(T, $l.s$, D, p_{from}, $l.m$)
            unicast(p_s,m')
        end for
    else if $m == \heartsuit$ **then** ▷ This is a heartbeat message.
        discard(m)
    else ▷ This is a regular message.
        $p_s = p_{from}$ ▷ Rewrite p_s in case of retransmission.
        if $s == D[p_s] + 1$ **then**
            incrementTimestamp(p_{self},T)
            if $T_l \not> q.T$ **then** ▷ Ensure causality of new message delivery.
                deliver(m)
                $T_l = T_m$
                $D[p_s] = s$
                $l.m = m$
                $l.s = s$
                addToMsgStore(l,$S[p_s]$)
            else
                $q.m = m$
                $q.s = s$
                $q.T = T_m$
                addToQ(q,$Q[p_s]$)
            end if

Algorithm 3 Reliable multicast receive (continued)

repeat ▷ Attempt to deliver items in queue.
for $q \in Q[p] \forall p \in G$ s.t. $q.s == D[p_s] + 1$ and $T_l \not> q.T$ **do**
deliver($q.m$)
$T_l = q.T$
$D[p_s] = q.s$
removeFromQ(Q,q)
$l.m = q.m$
$l.s = s$
addToMsgStore(l,$S[p]$)
end for
until Q unchanged.
else if $s > D[p_s] + 1$ **then** ▷ Message arrived out of order.
incrementTimestamp(p_{self},T)
$q.m = m$
$q.s = s$
$q.T = T_m$
addToQ(q,$Q[p_s]$)
else ▷ Message already delivered.
discard(m)
end if
end if
▷ Ask someone to re-transmit the messages that we don't have.
for $s' > D[p_{self}][p] \neq q.s \forall q \in Q[p], p \in G$ **do**
if $\exists s'' \geq s' \in D[p']$ for some $p_{self} \in G$ **then**
m' = piggyback(T, s'', $D[p_{self}]$, p, Ξ);
unicast(p_s,m') ▷ Ask p' to send us the message it got from p.
end if
end for
if unable to find anyone from which to request missing messages **then**
die()
end if
end procedure ▷ Updates G,Q,T,T_l, and D.