ECE 428 MP1 Design Documentation

Josiah McClurg and Yihua Lou

February 24, 2013

Abstract

1 Introduction

2 Methods

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 2 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 described by Algorithms 1, 2, and 3.

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 1 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 2.

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

(c) is ensured by allowing retransmissions from other processes to masquerade as the original sender.

Input: Multicast group G, own process identifier p_{self}, message m, sequence number s, along with vector timestamp T, and delivery acknowledgment set D all indexed by $p \in G$.
Output: Updates s, and S.
for *each* $p \neq p_{self} \in G$ **do**
    incrementTimestamp(p_{self},T);
    m' = piggyback(T, s, $D[p_{self}]$, p, m);
    unicast(p,m');
end
incrementSequenceNumber(s);

Algorithm 1: Reliable multicast send

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 3 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 2 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 3 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

Input: G,D,T, p_{self}, message source p_s, message m', along with delivered message store S, holdback queue Q, last delivery timestamp T_l, and timeout list L all indexed by $p \in G$.
Output: Updates G,Q,T,T_l, and D.
$\{T_m, s, D[p_s], p_{from}, m\}$ = unpiggyback(m'); mergeTimestamps(T,T_m);
$L[p_s]$ = time();
for *each* $p \in D[p_s]$ *such that* $p \notin D$ **do**
  removeFromGroup(p,G,Q,T,T_l,D);
end
for *each* $l \in S[p] \forall p \in G$ *such that* $l.s \leq min(D[p])$ **do**
  removeFromMsgStore(l,S);
end
if $m == \Xi$ **then**
  discard(m);
  for *each* $l \in S[p_{from}]$ *such that* $l.s \leq s$ **do**
    m' = piggyback(T, $l.s$, D, p_{from}, $l.m$);
    unicast(p_s,m');
  end
else if $m == \heartsuit$ **then**
  discard(m);
else
  $p_s = p_{from}$;
  if $s == D[p_s] + 1$ **then**
    incrementTimestamp(p_{self},T);
    if $T_l \not> q.T$ **then**
      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
    repeat
      for *each* $q \in Q[p] \forall p \in G$ *such that* $q.s == D[p_s] + 1$ *AND* $T_l \not> q.T$ **do**
        deliver($q.m$); removeFromQ(Q,q); $T_l = q.T$; $D[p_s] = q.s$;
        $l.m = q.m$; $l.s = s$; addToMsgStore(l,$S[p]$);
      end
    until Q *unchanged.*;
  else if $s > D[p_s] + 1$ **then**
    incrementTimestamp(p_{self},T);
    $q.m = m$; $q.s = s$; $q.T = T_m$; addToQ(q,$Q[p_s]$);
  else
    discard(m);
  end
end
for *each* $p \neq p' \in G$ *such that* $D[p_{self}][p] < D[p'][p]$ *AND* $D[p'][p] \neq q.s \forall q \in Q[p_s]$ **do**
  m' = piggyback(T, $D[p'][p]$, $D[p_{self}]$, p, Ξ); unicast(p_s,m');
end

Algorithm 2: Reliable multicast receive

Input: G, p_{self}, Q, T, T_l, D, and L.
Output: Updates G,Q,T,T_l, and D.
repeat
    for *each* $p \neq p_{self} \in G$ **do**
        if $\mathit{time}() - L[p] \geq T_f$ **then**
            removeFromGroup(p,G,Q,T,T_l,D);
        else
            if $\mathit{time}() - L[p_{self}] \geq T_h$ **then**
                m' = piggyback(T, 0, $D[p_{self}]$, p, $\heartsuit$);
                unicast(p,m');
            end
        end
    end
    $t = T_h -$ (time() $- L[p_{self}]$);
    sleep(min(t,0));
until *end of program*;

Algorithm 3: Failure detect thread.