Reliable Multicast Protocol Specifications Protocol Operation

by John R. Callahan, Todd Montgomery, and Brian Whetten
This appendix contains the complete state tables for RMP Normal Operation, Multi-RPC Extensions, Membership Change Extensions, and Reformation Extensions. First the event types are presented. Afterwards, each RMP operation state, normal and extended is presented individually and its events shown.

Events in the RMP specification are one of several things. (1) Arriving packets, (2) Expired alarms, (3) User events, (4) Exceptional conditions. The specification event types are:

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Data Packet</td>
</tr>
<tr>
<td>ACK</td>
<td>ACK Packet</td>
</tr>
<tr>
<td>NACK</td>
<td>NACK Packet</td>
</tr>
<tr>
<td>Conf</td>
<td>Confirm Packet</td>
</tr>
<tr>
<td>NMD</td>
<td>Non-Member Data Packet</td>
</tr>
<tr>
<td>NMA</td>
<td>Non-Member ACK Packet</td>
</tr>
<tr>
<td>NL</td>
<td>New List Packet</td>
</tr>
<tr>
<td>LCR</td>
<td>List Change Request Packet</td>
</tr>
<tr>
<td>RecStart</td>
<td>Recovery Start Packet</td>
</tr>
<tr>
<td>RecVote</td>
<td>Recovery Vote Packet</td>
</tr>
<tr>
<td>RecACKNL</td>
<td>Recovery ACK New List Packet</td>
</tr>
<tr>
<td>RecAbort</td>
<td>Recovery Abort Packet</td>
</tr>
<tr>
<td>Failure</td>
<td>Retransmission timeout on packet</td>
</tr>
<tr>
<td>TPA</td>
<td>Token Pass Alarm</td>
</tr>
<tr>
<td>CTPA</td>
<td>Confirm Token Pass Alarm</td>
</tr>
<tr>
<td>RTA</td>
<td>Random Timeout Alarm</td>
</tr>
<tr>
<td>MandLv</td>
<td>Mandatory Leave Alarm</td>
</tr>
<tr>
<td>CommitNL</td>
<td>Commit New List Notification</td>
</tr>
<tr>
<td>JoinReq</td>
<td>Application request to join group</td>
</tr>
</tbody>
</table>

Packet Positive Acknowledgements and Fault Detection

Some packets are retransmitted on a scheduled basis until a given set of condition occur that cease that retransmit schedule. This set of conditions can be considered as a positive acknowledgement for the
The different RMP packets and the positive acknowledgement conditions for each are shown in the table below. The packets with (none) as their positive acknowledgement conditions are only sent once and are not scheduled for any kind of retransmission.

<table>
<thead>
<tr>
<th>Packet Type</th>
<th>Positive Acknowledgment Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Reception of ACK Packet that contains packet</td>
</tr>
<tr>
<td>ACK</td>
<td>Reception of ACK, New List, or Confirm packet with Timestamp &gt;= Timestamp of packet</td>
</tr>
<tr>
<td>NACK</td>
<td>Reception of requested Packet(s)</td>
</tr>
<tr>
<td>Confirm</td>
<td>(none)</td>
</tr>
<tr>
<td>Non-Member Data (NMD)</td>
<td>Reception of Non-Member ACK Packet that contains packet</td>
</tr>
<tr>
<td>(none)</td>
<td></td>
</tr>
<tr>
<td>New List (NL)</td>
<td>Reception of ACK, New List, or Confirm packet with Timestamp &gt;= Timestamp of packet</td>
</tr>
<tr>
<td>List Change Request (LCR)</td>
<td>Reception of a New List Packet containing response to request</td>
</tr>
<tr>
<td>Recovery Start</td>
<td>Creation of a valid New List Packet or X retransmits</td>
</tr>
<tr>
<td>Recovery Vote</td>
<td>Reception of a New List Packet from Reform Site</td>
</tr>
<tr>
<td>Recovery ACK New List</td>
<td>Reception of Null ACK from Reform Site</td>
</tr>
<tr>
<td>Recovery Abort</td>
<td>(none)</td>
</tr>
</tbody>
</table>

If a set number of retransmissions, call this value X, for a packet occur without the positive acknowledgement conditions being met, then a Failure event is generated for that packet. This event is indicative of a failure (or blockage) being detected within the protocol operation (the cause of which being an application failure, a site failure, or a communication failure). Failures in RMP are assumed to be failures that are non-corruptive. All members are assumed to be well behaved in the presence of failures and not miscreant.

Duplicate Detection and Filtering of Packets

Packets must pass through two devices before they are allowed to be processed. The first level is filtering. This level examines the packet type and TRID. If the TRID is not a valid TRID, then the packet is dropped. The state of the site is also taken into account. When the site is in the Not In Ring state or the Joining Ring state, it does not have information about the current TRID of the group. Thus filtering must be done based on packet type and packet contents. For example, as specified below, a site in the Joining Ring state transitions into the Token Site state upon a New List packet. To
filter out possible errant transitions, the filter must drop all other New List packets not meeting the transition criteria.

The second device is duplicate detection. Duplicates are assumed to be dropped and not processed. Duplicates can be detected based on the packet type and state of the site. For packets with explicit timestamps, ACKs and NLs, the timestamp can be used to determine if the packet is a duplicate. For implicit timestamped packets, Data, NMD, and LCRs, the job is slightly more complicated. Careful track of sequence numbers from clients as well as group members must be kept for duplicates to be detected. NMA, NACKs and Confirm packets are never checked for being duplicates. During reformation, duplicates are allowed to be processed (as long as they are part of reformation. Data, NMD, ACK, NL, and LCR packets are still put through duplicate detection). Another exception to the duplicate detection rule is when the site is in the Token Site state. Duplicate detection is not done on any ACK or NL packets that arrive when the site is in the Token Site state.

Algorithms and Data Structures

Two data structures are used by RMP. The first is the DataQ. This is a FIFO structure that holds Data, NMD, and LCR packets as they arrive. The second structure is an Ordered FIFO called an OrderingQ. This FIFO is ordered based on timestamps. Each member of the FIFO, called a slot, is assigned a unique timestamp. In this way, the FIFO has elements which are monotonically increasing. Each slot of the OrderingQ is one of the following types of packets: Data, NMD, ACK, or NL. ACK and NL packets contain explicit timestamps to order them in the OrderingQ. Data, NMD, and LCR packets are given implicit timestamps by the corresponding ACK or NL that addresses them. Therefore, an ACK with timestamp of 5 which acknowledges 3 packets will implicitly give 3 Data or NMD packets implicit timestamps of 6, 7, and 8. LCR packets are dropped once a corresponding NL is received. LCRs are never put into the OrderingQ.

In the state tables given below, it is assumed that the addition of an ACK into the OrderingQ also enqueues empty slots for each packet acknowledged by the ACK. Each slot in the OrderingQ is assigned a state. There are four possible states for each slot. They are: (1) Packet Missing, (2) Packet Requested, (3) Packet Received, and (4) Packet Delivered. A slot usually starts out as being in the Packet Missing state. Once a NACK is sent for that packet, the state changes to Packet Requested. When the slot is filled by a packet, the state changes to Packet Received, and, finally, when the packet is delivered to the application the slot state is changed to Packet Delivered.
A few algorithms are provided to specify the operation of the OrderingQ and DataQ during the protocol operation. The first algorithm is the Update-OrderingQ algorithm presented below.

Update-OrderingQ():
for each (slot in the OrderingQ (starting with lowest timestamp))
    If (slot timestamp not equal to last slot timestamp + 1) then
        EnQueue as many empty slots to cover missing timestamps
        for each (new slot to be Enqueued) Loop
            Send NACK for missing timestamp
            Mark slot state as Packet Requested
        End for.
    End If.
    If (slot state is Packet Missing) then
        Search DataQ for missing packet
        If (packet is found in DataQ) then
            Remove packet from DataQ
            Place packet in OrderingQ
            Mark slot as Packet Received
            Attempt-Packet-Delivery(slot)
            Update information about packet source
        Else
            Send NACK for packet
            Mark slot as Packet Requested
        End If.
    Else If (slot state is Packet Committed) then
        If (token has been passed enough times to satisfy QoS resiliency requirements) then
            Mark slot as Packet Delivered
            Notify application that QoS resiliency has been met
            Update information about packet source
        End If.
    Else If (slot state is Packet Requested) then
        Search DataQ for missing packet
        If (packet is found in DataQ) then
            Remove packet from DataQ
            Place packet in OrderingQ
            Mark slot as Packet Received
            Attempt-Packet-Delivery(slot)
            Update information about packet source
        End If.
        Else If (slot state is Packet Received) then
            Attempt-Packet-Delivery(slot)
            Update information about packet source
        Else If (slot state is Packet Delivered) then
            Update information about packet source
        End If.
    End If.
End for.
while (the number of ACK Packets and New Lists Packets in OrderingQ is greater than the number of members of the Token Ring) Loop
  DeQueue lowest timestamp and discard packet
End while.
End Update-OrderingQ.

As auxilliary algorithm, Attempt-Packet-Delivery, is used by Update-OrderingQ to determine if a packet is deliverable and to deliver the packet to the application. The Attempt-Packet-Delivery algorithm is presented below.

Attempt-Packet-Delivery(slot):
  If (slot is a Data Packet or Non-Member Data Packet) then
    If (slot packet has QoS equal to Unordered) then
      Commit the packet to the application
      Mark slot as Packet Delivered
    Else If (slot packet has QoS equal to Source Ordered) then
      If (all of the smaller sequence numbers from that source have been delivered) then
        Commit the packet to the application
        Mark slot as Packet Delivered
      End If.
    Else If (slot packet has QoS equal to Totally Ordered) then
      If (all of the timestamps smaller than the slots timestamps have been delivered) then
        Commit the packet to the application
        Mark slot as Packet Delivered
      End If.
    Else If (slot packet has QoS equal to K Resilient or slot packet has QoS equal to Majority Resilient or slot packet has QoS equal to Totally Resilient) then
      If (all of the timestamps smaller than the slots timestamps have been delivered) then
        Commit the packet to the application
        Mark slot as Packet Committed
      End If.
  End If.
  Else if (slot is a New List Packet) then
    If (all of the timestamps smaller than the slots timestamps have been delivered) then
      Commit the New List and notify application
      Mark slot as Packet Delivered
    End If.
  Else if (slot is an ACK Packet) then
    Mark slot as Packet Delivered
  End If.
End Attempt-Packet-Delivery.
These two algorithms describe the behavior desired for packets to be ordered and delivered. The last algorithm presented is the Pass-Token algorithm. This algorithm describes how a Token Site is to fill pass the token through either a NL or an ACK. The algorithm is presented below.

Pass-Token():
for each (member of the DataQ)
    If (member is a List Change Request Packet and request can be granted and packet is eligible) then
        Generate a New List Packet for request
        Send New List Packet
        Exit Loop
    Else If (member is a Data Packet or a Non-Member Data Packet and is eligible to be acknowledged) then
        Generate ACK Packet containing as many Data Packets and Non-Member Data Packets as are eligible in the DataQ
        Send ACK Packet
        Exit Loop
    End If.
End for.
If (ACK Packet or New List Packet could not be generated) then
    Return to calling routine reporting Token Not Passed
Else
    Return to calling routine reporting Token Passed
End If.
End Pass-Token.

For an LCR, Data, or NMD packet to be eligible, the sequence number of the packet must have the expected next value for the source. No strict upper bound is placed on the number of NMD or Data packets an ACK may contain, although differing implementations may impose their own limits.

Delivery of low QoS packets is also checked as the packet arrives and placed into the DataQ. If conditions are such that the packet meets its QoS when it arrives, then the packet may be delivered as well as being placed into the DataQ. Thus when the Update-OrderingQ algorithm pulls the packet out of the DataQ, the slot into which it is placed is to be marked as Packet Delivered instead of Packet Received. This behavior is assumed in the state tables presented below. Another behavior assumed in the preceeding algorithms is that the implementation will block packets with resilient QoSs from actual delivery to the application until the conditions meeting their QoS are met. This does mean blocking the delivery of other QoSs (Source and Totally Ordered) that depend on that higher QoS packet.

State Tables and Actions
In the tables presented below each state is shown along with the actions and conditions required to transition into the next state. Assume that events not shown are ignored unless they are specifically discussed below. The different states of RMP operation are:

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS</td>
<td>Token Site State</td>
</tr>
<tr>
<td>NTS</td>
<td>Not Token Site State</td>
</tr>
<tr>
<td>GP</td>
<td>Getting Packets State</td>
</tr>
<tr>
<td>PT</td>
<td>Passing Token State</td>
</tr>
<tr>
<td>JR</td>
<td>Joining Ring State</td>
</tr>
<tr>
<td>LR</td>
<td>Leaving Ring State</td>
</tr>
<tr>
<td>NIR</td>
<td>Not In Ring State</td>
</tr>
<tr>
<td>SR</td>
<td>Start Recovery State</td>
</tr>
<tr>
<td>CNL</td>
<td>Created New List State</td>
</tr>
<tr>
<td>SV</td>
<td>Sent Vote State</td>
</tr>
<tr>
<td>ACKNL</td>
<td>ACK New List State</td>
</tr>
<tr>
<td>AR</td>
<td>Abort Recovery State</td>
</tr>
</tbody>
</table>

In all states the following behaviors are necessary. All NACK events are handled the same way no matter what the current state is. The default action for NACK responses is to examine the OrderingQ and retransmit the requested timestamped packet (either explicit timestamp or implicit timestamp). NACK policy controls how these are sent, via multicast or unicast, and the timing factors involved. Another issue is generation of NMA packets in response to NMD packets. The default behavior is to have the site which generates the ACK containing the NMD to generate an NMA for that NMD and unicast the NMA to the client. If after this is done, a member of the group sees a duplicate NMD (one that has already been acknowledged and ordered), then the member of the list generates an NMA and unicasts it to the client. The down side of this is that this may result in NMA implosion to the client. This policy may also be changed on an implementation basis to be that the same site always sends the NMA.

It is recommended that implementations differentiate, to the application, NMA packets that hold replies and NMA packets that do not hold replies. The default behavior for a client is to periodically send an NMD to the group (via unicast or multicast) until it receives an NMA (holding no reply) for that NMD, notify the application that the group has received the message, then wait for another NMA that holds a reply. If the reply is in the form of multiple NMA packets, then each should be delivered as they arrive. Missing NMA replies can be requested by sending another NMD (with an increased sequence number), thus causing another message to be delivered to the group. This can be done repeatedly until a client finally gets its reply. Alternatively, a client may mark the Multiple Copies flag of the NMD so that multiple copies of the same NMD will be delivered to the application as they arrive.
Due to the fact that this set of policies does not give any guarantees to the client that the desired QoS was met, the application is fully responsible for the response policy that it will use. For example, if clients built for an application need to know when the QoS is met for the NMD, then the group must send a NMA with a response when that QoS is met and the NMD delivered to the application.
Token Site State Table (current state is TS). Events not mentioned have no action and cause no transition. The sequence of actions are executed before the conditions are checked.

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition(s)</th>
<th>Next State</th>
<th>Action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Token Passed</td>
<td>PT</td>
<td>place packet in DataQ, Pass-Token</td>
</tr>
<tr>
<td>Data</td>
<td>!Token Passed</td>
<td>TS</td>
<td>place packet in DataQ, Pass-Token</td>
</tr>
<tr>
<td>NMD</td>
<td>Token Passed</td>
<td>PT</td>
<td>place packet in DataQ, Pass-Token</td>
</tr>
<tr>
<td>NMD</td>
<td>!Token Passed</td>
<td>TS</td>
<td>place packet in DataQ, Pass-Token</td>
</tr>
<tr>
<td>LCR</td>
<td>Token Passed</td>
<td>PT</td>
<td>place packet in DataQ, Pass-Token</td>
</tr>
<tr>
<td>LCR</td>
<td>!Token Passed</td>
<td>TS</td>
<td>place packet in DataQ, Pass-Token</td>
</tr>
<tr>
<td>ACK</td>
<td>Named Token</td>
<td>TS</td>
<td>Unicast Confirm to Source</td>
</tr>
<tr>
<td>NL</td>
<td>Named Token</td>
<td>TS</td>
<td>Unicast Confirm to Source</td>
</tr>
<tr>
<td>Failure</td>
<td>(none)</td>
<td>SR</td>
<td>Multicast RecStart</td>
</tr>
<tr>
<td>RecStart</td>
<td>(none)</td>
<td>SV</td>
<td>Unicast RecVote to Reform Site</td>
</tr>
<tr>
<td>TPA</td>
<td>(none)</td>
<td>PT</td>
<td>Generate Null ACK, Multicast Null ACK</td>
</tr>
<tr>
<td>CTFA</td>
<td>(none)</td>
<td>TS</td>
<td>Unicast Confirm to last Token Site</td>
</tr>
</tbody>
</table>
Passing Token State Table (current state is PT). Events not mentioned have no action and cause no transition. The sequence of actions are executed _before_ the conditions are checked.

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition(s)</th>
<th>Next State</th>
<th>Action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>(none)</td>
<td>PT</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>NMD</td>
<td>(none)</td>
<td>PT</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>LCR</td>
<td>(none)</td>
<td>PT</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>NL</td>
<td>!named Token Site</td>
<td>NTS</td>
<td>Add NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>NL</td>
<td>named Token Site</td>
<td>PT</td>
<td>Add NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>OrderingQ consistent</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td>Token passed</td>
<td></td>
<td>Pass-Token</td>
</tr>
<tr>
<td>NL</td>
<td>named Token Site</td>
<td>TS</td>
<td>Add NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>OrderingQ consistent</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td>!Token passed</td>
<td></td>
<td>Pass-Token</td>
</tr>
<tr>
<td>NL</td>
<td>named Token Site</td>
<td>GP</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>!OrderingQ consistent</td>
<td>NTS</td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>ACK</td>
<td>!named Token Site</td>
<td></td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>ACK</td>
<td>named Token Site</td>
<td>PT</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>OrderingQ consistent</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td>Token passed</td>
<td></td>
<td>Pass-Token</td>
</tr>
<tr>
<td>ACK</td>
<td>named Token Site</td>
<td>TS</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>OrderingQ consistent</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td>!Token passed</td>
<td></td>
<td>Pass-Token</td>
</tr>
<tr>
<td>ACK</td>
<td>named Token Site</td>
<td>GP</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>!OrderingQ consistent</td>
<td>NTS</td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>Conf</td>
<td>Timestamp &gt;=</td>
<td>NTS</td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td>Last token pass</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Timestamp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Failure</td>
<td>(none)</td>
<td>SR</td>
<td>Multicast RecStart</td>
</tr>
<tr>
<td>RecStart</td>
<td>(none)</td>
<td>SV</td>
<td>Unicast RecVote to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reform Site</td>
</tr>
</tbody>
</table>
Not Token Site State Table (current state is NTS). Events not mentioned have no action and cause no transition. The sequence of actions are executed _before_ the conditions are checked.

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition(s)</th>
<th>Next State</th>
<th>Action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>(none)</td>
<td>NTS</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>NMD</td>
<td>(none)</td>
<td>NTS</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>LCR</td>
<td>(none)</td>
<td>NTS</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>NL</td>
<td>!named Token Site</td>
<td>NTS</td>
<td>Add NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>NL</td>
<td>named Token Site</td>
<td>PT</td>
<td>Add NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>OrderingQ consistent</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td>Token passed</td>
<td></td>
<td>Pass-Token</td>
</tr>
<tr>
<td>NL</td>
<td>named Token Site</td>
<td>TS</td>
<td>Add NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>OrderingQ consistent</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td>!Token passed</td>
<td></td>
<td>Pass-Token</td>
</tr>
<tr>
<td>NL</td>
<td>named Token Site</td>
<td>GP</td>
<td>Add NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>!OrderingQ consistent</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>ACK</td>
<td>!named Token Site</td>
<td>NTS</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>ACK</td>
<td>named Token Site</td>
<td>PT</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>OrderingQ consistent</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td>Token passed</td>
<td></td>
<td>Pass-Token</td>
</tr>
<tr>
<td>ACK</td>
<td>named Token Site</td>
<td>TS</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>OrderingQ consistent</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td>!Token passed</td>
<td></td>
<td>Pass-Token</td>
</tr>
<tr>
<td>ACK</td>
<td>named Token Site</td>
<td>GP</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>!OrderingQ consistent</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>Failure</td>
<td>(none)</td>
<td>SR</td>
<td>Multicast RecStart</td>
</tr>
<tr>
<td>RecStart</td>
<td>(none)</td>
<td>SV</td>
<td>Unicast RecVote to Reform Site</td>
</tr>
<tr>
<td>CommitNL</td>
<td>NL does not contain site</td>
<td>LR</td>
<td>Schedule MandLv</td>
</tr>
</tbody>
</table>
Getting Packets State Table (current state is GP). Events not mentioned have no action and cause no transition. The sequence of actions are executed _before_ the conditions are checked.

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition(s)</th>
<th>Next State</th>
<th>Action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>OrderingQ consistent</td>
<td>PT</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td>Token passed</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>Data</td>
<td>OrderingQ consistent</td>
<td>TS</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td>!Token passed</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>Data</td>
<td>!OrderingQ consistent</td>
<td>GP</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td>Token passed</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>NMD</td>
<td>OrderingQ consistent</td>
<td>PT</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td>Token passed</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>NMD</td>
<td>!OrderingQ consistent</td>
<td>TS</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td>!Token passed</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>LCR</td>
<td>(none)</td>
<td>GP</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td>ACK</td>
<td>OrderingQ consistent</td>
<td>PT</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>Token passed</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>ACK</td>
<td>!OrderingQ consistent</td>
<td>TS</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>!Token passed</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>ACK</td>
<td>!OrderingQ consistent</td>
<td>GP</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td>NL</td>
<td>OrderingQ consistent</td>
<td>PT</td>
<td>Add NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>Token passed</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>NL</td>
<td>!OrderingQ consistent</td>
<td>TS</td>
<td>Add NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>!Token passed</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>Failure</td>
<td>(none)</td>
<td>SR</td>
<td>Multicast RecStart</td>
</tr>
<tr>
<td>RecStart</td>
<td>(none)</td>
<td>SV</td>
<td>Unicast RecVote to Reform Site</td>
</tr>
</tbody>
</table>
Not In Ring State Table (current state is NIR). Events not mentioned have no action and cause no transition. The sequence of actions are executed _before_ the conditions are checked.

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition(s)</th>
<th>State</th>
<th>Action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JoinReq</td>
<td>(none)</td>
<td>JR</td>
<td>Multicast LCR to join ring</td>
</tr>
<tr>
<td>NMA</td>
<td>NMA holds reply</td>
<td>NIR</td>
<td>Deliver reply to application</td>
</tr>
</tbody>
</table>

Joining Ring State Table (current state is JR). Events not mentioned have no action and cause no transition. The sequence of actions are executed _before_ the conditions are checked.

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition(s)</th>
<th>State</th>
<th>Action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>named Token Site</td>
<td>TS</td>
<td>Add NL to OrderingQ</td>
</tr>
<tr>
<td>Failure</td>
<td>(none)</td>
<td>TS</td>
<td>Generate NL to form own Token Ring</td>
</tr>
</tbody>
</table>

Leaving Ring State Table (current state is LR). Events not mentioned have no action and cause no transition. The sequence of actions are executed _before_ the conditions are checked.

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition(s)</th>
<th>State</th>
<th>Action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>Exit condition met</td>
<td>NIR</td>
<td>(none)</td>
</tr>
<tr>
<td>NL</td>
<td>!Exit condition met</td>
<td>LR</td>
<td>(none)</td>
</tr>
<tr>
<td>ACK</td>
<td>Exit condition met</td>
<td>NIR</td>
<td>(none)</td>
</tr>
<tr>
<td>ACK</td>
<td>!Exit condition met</td>
<td>LR</td>
<td>(none)</td>
</tr>
<tr>
<td>MandLv</td>
<td>(none)</td>
<td>NIR</td>
<td>(none)</td>
</tr>
</tbody>
</table>

NOTE: Exit condition is that a number of Token Passes has occurred in the group equal to the number of people in the group when the site transitioned into LR.
Start Recovery State Table (current state is SR). Events not mentioned have no action and cause no transition. The sequence of actions are executed _before_ the conditions are checked.

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition(s)</th>
<th>Next State</th>
<th>Action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>(none)</td>
<td>SR</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update SynchTSP</td>
</tr>
<tr>
<td>NMD</td>
<td>(none)</td>
<td>SR</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update SynchTSP</td>
</tr>
<tr>
<td>LCR</td>
<td>(none)</td>
<td>SR</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update SynchTSP</td>
</tr>
<tr>
<td>ACK</td>
<td>(none)</td>
<td>SR</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update SynchTSP and MaxTSP</td>
</tr>
<tr>
<td>NL</td>
<td>(none)</td>
<td>SR</td>
<td>Add NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update SynchTSP and MaxTSP</td>
</tr>
<tr>
<td>Failure</td>
<td>packet is</td>
<td>CNL</td>
<td>Create New List</td>
</tr>
<tr>
<td></td>
<td>RecStart</td>
<td></td>
<td>Multicast New List</td>
</tr>
<tr>
<td></td>
<td>!only site</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Failure</td>
<td>packet is</td>
<td>NIR</td>
<td>Create New List</td>
</tr>
<tr>
<td></td>
<td>RecStart</td>
<td></td>
<td>Commit New List</td>
</tr>
<tr>
<td></td>
<td>only site</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>List is invalid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Failure</td>
<td>packet is</td>
<td>TS</td>
<td>Create New List</td>
</tr>
<tr>
<td></td>
<td>RecStart</td>
<td></td>
<td>Commit New List</td>
</tr>
<tr>
<td></td>
<td>only site</td>
<td></td>
<td>Multicast New List</td>
</tr>
<tr>
<td></td>
<td>List is valid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RecVote</td>
<td>incorrect Info</td>
<td>AR</td>
<td>Multicast RecAbort</td>
</tr>
<tr>
<td>RecVote</td>
<td>Vote MaxTSP &gt; MaxTSP</td>
<td>SR</td>
<td>Update MaxTSP</td>
</tr>
<tr>
<td>RecVote</td>
<td>Vote MaxTSP &lt;= MaxTSP</td>
<td>SR</td>
<td>Update source vote</td>
</tr>
<tr>
<td>RecVote</td>
<td>OrderingQ consistent</td>
<td>CNL</td>
<td>Create New List</td>
</tr>
<tr>
<td></td>
<td>Vote from each site</td>
<td></td>
<td>Multicast New List</td>
</tr>
<tr>
<td></td>
<td>Each Voter synched</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTE: incorrect Info means that the Token Ring Version, New Token Ring ID, or Reform Site information in the packet is incorrect. Any update to the value of MaxTSP or SynchTSP for the Reform Site prompts a restart of the RecStart packet. Each Voter is synched if the vote from that voter has its SynchTSP set to the MaxTSP of the current reformation.
Created New List State Table (current state is CNL). Events not mentioned have no action and cause no transition. The sequence of actions are executed _before_ the conditions are checked.

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition(s)</th>
<th>Next State</th>
<th>Action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>(none)</td>
<td>CNL</td>
<td>place packet in DataQ Update-OrderingQ</td>
</tr>
<tr>
<td>NMD</td>
<td>(none)</td>
<td>CNL</td>
<td>place packet in DataQ Update-OrderingQ</td>
</tr>
<tr>
<td>LCR</td>
<td>(none)</td>
<td>CNL</td>
<td>place packet in DataQ Update-OrderingQ</td>
</tr>
<tr>
<td>RecACKNL</td>
<td>incorrect Info</td>
<td>AR</td>
<td>Multicast RecAbort</td>
</tr>
<tr>
<td>RecACKNL</td>
<td>!All sites have ACKed</td>
<td>CNL</td>
<td>Mark source as ACKed</td>
</tr>
<tr>
<td>RecACKNL</td>
<td>All sites have ACKed List is valid</td>
<td>PT</td>
<td>Add NL to OrderingQ Commit NL</td>
</tr>
<tr>
<td>RecACKNL</td>
<td>All sites have ACKed List is invalid</td>
<td>NIR</td>
<td>Add NL to OrderingQ Commit NL</td>
</tr>
<tr>
<td>Failure</td>
<td>packet is NL List is valid</td>
<td>AR</td>
<td>Multicast RecAbort</td>
</tr>
<tr>
<td>Failure</td>
<td>packet is NL List is invalid</td>
<td>NIR</td>
<td>Add NL to OrderingQ Commit NL</td>
</tr>
<tr>
<td>RecAbort</td>
<td>correct Info</td>
<td>AR</td>
<td>Multicast RecAbort</td>
</tr>
<tr>
<td>RecStart</td>
<td>incorrect Info</td>
<td>AR</td>
<td>Unicast Reformation NL to voting site</td>
</tr>
<tr>
<td>RecVote</td>
<td>correct Info</td>
<td>CNL</td>
<td>Unicast Reformation</td>
</tr>
</tbody>
</table>

NOTE: Reformation NL is generated in the SR state. This NL is not added to the OrderingQ or committed by the Reform Site until it is positive that all the Slave Sites have it. This is shown as having ACKs from all those involved in Reformation. Valid lists are lists that pass the Minimum Size Requirements as set forth by the members. Invalid lists fail to meet these requirements.

ACK and NL packets that are not duplicates arriving in this state are to cause warnings. The ring should not be sending them because the synchronization point for the group has already been chosen and should not be changed.
Sent Vote State Table (current state is SV). Events not mentioned have no action and cause no transition. The sequence of actions are executed before the conditions are checked.

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition(s)</th>
<th>State</th>
<th>Next Action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>(none)</td>
<td>SV</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update RecVote</td>
</tr>
<tr>
<td>NMD</td>
<td>(none)</td>
<td>SV</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update RecVote</td>
</tr>
<tr>
<td>LCR</td>
<td>(none)</td>
<td>SV</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update RecVote</td>
</tr>
<tr>
<td>ACK</td>
<td>(none)</td>
<td>SV</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update RecVote</td>
</tr>
<tr>
<td>NL</td>
<td>!Reformation NL</td>
<td>SV</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update RecVote</td>
</tr>
<tr>
<td>NL</td>
<td>Reformation NL</td>
<td>NIR</td>
<td>Unicast RecACKNL to Reform Site</td>
</tr>
<tr>
<td></td>
<td>!site in list</td>
<td></td>
<td>Commit NL</td>
</tr>
<tr>
<td>NL</td>
<td>Reformation NL</td>
<td>NIR</td>
<td>Unicast RecACKNL to Reform Site</td>
</tr>
<tr>
<td></td>
<td>List is invalid</td>
<td></td>
<td>Commit NL</td>
</tr>
<tr>
<td>NL</td>
<td>Reformation NL</td>
<td>ACKNL</td>
<td>Unicast RecACKNL to Reform Site</td>
</tr>
<tr>
<td></td>
<td>List is valid</td>
<td></td>
<td>Commit NL</td>
</tr>
<tr>
<td>RecStart</td>
<td>correct Info</td>
<td>SV</td>
<td>Unicast RecVote to Reform Site</td>
</tr>
<tr>
<td>Failure</td>
<td>incorrect Info</td>
<td>AR</td>
<td>Multicast RecAbort</td>
</tr>
<tr>
<td>RecAbort</td>
<td>correct Info</td>
<td>AR</td>
<td>Multicast RecAbort</td>
</tr>
</tbody>
</table>

NOTE: Each time a Slave Site receives a RecStart from the Reform Site, the Slave Site is to restart its retransmission schedule for its RecVote. This is needed so that updates to the MaxTSP and SynchTSP of the reformation do not cause a Slave Site to prematurely initiate abortion of a reformation. Every time a Slave Site updates its RecVote, the site must also restart its retransmit schedule on the RecVote packet. A Reformation NL is a New List that meets the following criteria: (1) Version number is correct with current Reformation, (2) The current Token Site and the next Token Site are the current Reform Site, (3) The Timestamp of the NL must be in the range SynchTSP+1 to MaxTSP+1 of the current Reformation, (4) The operation type of the NL must be a reformation operation, and (5) The new TRID of the NL must be equal to the TRID of the reformation.

It is practically impossible at this step of reformation to tell the
difference between a viable reformation NL and an incorrect one.
ACK New List State Table (current state is ACKNL). Events not mentioned have no action and cause no transition. The sequence of actions are executed _before_ the conditions are checked.

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition(s)</th>
<th>Next State</th>
<th>Action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>(none)</td>
<td>ACKNL</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td>NMD</td>
<td>(none)</td>
<td>ACKNL</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td>LCR</td>
<td>(none)</td>
<td>ACKNL</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td>NL</td>
<td>Timestamp &lt;</td>
<td>ACKNL</td>
<td>Add NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>Reformation NL</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td></td>
<td>NL</td>
<td>ACKNL</td>
<td>Unicast RecACKNL to Reform Site</td>
</tr>
<tr>
<td>NL</td>
<td>!Reformation NL incorrect Info</td>
<td>AR</td>
<td>Multicast RecAbort</td>
</tr>
<tr>
<td></td>
<td>Reformation NL &gt; Timestamp</td>
<td>NTS</td>
<td>Add Reformation NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>!named Token Site</td>
<td></td>
<td>Commit Reformation NL Add NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>Reformation NL</td>
<td>PT</td>
<td>Add Reformation NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>named Token Site</td>
<td></td>
<td>Commit Reformation NL Add NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>OrderingQ consistent</td>
<td></td>
<td>Update-OrderingQ Pass-Token</td>
</tr>
<tr>
<td></td>
<td>Token passed</td>
<td>TS</td>
<td>Add Reformation NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>Reformation NL &gt; Timestamp</td>
<td>GP</td>
<td>Add Reformation NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>!named Token Site</td>
<td></td>
<td>Commit Reformation NL Add NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>OrderingQ consistent</td>
<td></td>
<td>Update-OrderingQ Pass-Token</td>
</tr>
<tr>
<td>ACK</td>
<td>Timestamp &lt;</td>
<td>ACKNL</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>Reformation NL</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>ACK</td>
<td>Timestamp &gt;</td>
<td>NTS</td>
<td>Add Reformation NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>Reformation NL</td>
<td></td>
<td>Commit Reformation NL Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td>!named Token Site</td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>ACK</td>
<td>Timestamp &gt;</td>
<td>PT</td>
<td>Add Reformation NL</td>
</tr>
</tbody>
</table>
RMP Operation

Reliable Multicast Protocol

5 October 1995

Reformation NL
named Token Site
OrderingQ consistent
Token passed

ACK
Timestamp > TS
Reformation NL
named Token Site
OrderingQ consistent
Token passed

ACK
Timestamp > GP
Reformation NL
named Token Site
OrderingQ consistent

Failure
RecAbort correct Info AR
RecStart incorrect Info AR

NOTE:

Add Reformation NL to OrderingQ
Commit Reformation NL
Add ACK to OrderingQ
Update-OrderingQ
Pass-Token

Add Reformation NL to OrderingQ
Commit Reformation NL
Add ACK to OrderingQ
Update-OrderingQ
Pass-Token

Multicast RecAbort
Multicast RecAbort (none)

Whetten, Montgomery, Callahan RMP 1.3b [Page 19]
Abort Recovery State Table (current state is AR). Events not mentioned have no action and cause no transition. The sequence of actions are executed _before_ the conditions are checked.

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition(s)</th>
<th>Next State</th>
<th>Action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>(none)</td>
<td>AR</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>NMD</td>
<td>(none)</td>
<td>AR</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>LCR</td>
<td>(none)</td>
<td>AR</td>
<td>place packet in DataQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>NL</td>
<td>Old Version</td>
<td>AR</td>
<td>Add NL to OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>NL</td>
<td>Reformation NL</td>
<td>AR</td>
<td>Multicast RecAbort</td>
</tr>
<tr>
<td>Version is of last attempt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACK</td>
<td>(none)</td>
<td>AR</td>
<td>Add ACK to OrderingQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update-OrderingQ</td>
</tr>
<tr>
<td>RTA</td>
<td>(none)</td>
<td>SR</td>
<td>Multicast RecStart</td>
</tr>
<tr>
<td>RecStart</td>
<td>correct Version</td>
<td>SV</td>
<td>Unicast RecVote to Reform Site</td>
</tr>
<tr>
<td>RecStart</td>
<td>incorrect Version</td>
<td>AR</td>
<td>Multicast RecAbort</td>
</tr>
<tr>
<td>RecVote</td>
<td>incorrect Version</td>
<td>AR</td>
<td>Multicast RecAbort</td>
</tr>
<tr>
<td>RecACKNL</td>
<td>incorrect Version</td>
<td>AR</td>
<td>Multicast RecAbort</td>
</tr>
</tbody>
</table>

NOTE: A correct version at this stage is a version greater than the last known version. Fill the RecAbort packet in with data in the incoming packet. A NL that qualifies as a Reformation NL is any NL that has an operation type dealing with Reformation and has a version higher than the last known version.

Authors’ Addresses:

Brian Whetten  
University of California Berkeley  
Berkeley, CA  
Email: whetten@tenet.cs.berkeley.edu

Todd Montgomery  
West Virginia University  
Morgantown, WV  
Email: tmont@cerc.wvu.edu

John R. Callahan  
West Virginia University  
Morgantown, WV