Figure 11.1 A network partition

Crashed router
Figure 11.2  Server managing a mutual exclusion token for a set of processes

Queue of requests

1. Request token

2. Release token

3. Grant token

Server

$p_1$

$p_2$

$p_3$

$p_4$
Figure 11.3  A ring of processes transferring a mutual exclusion token
Figure 11.4 Ricart and Agrawala’s algorithm

On initialization
\[ state := \text{RELEASED}; \]

To enter the section
\[ state := \text{WANTED}; \]
Multicast request to all processes;
\[ T := \text{request’s timestamp}; \]
Wait until (number of replies received = \(N - 1\));
\[ state := \text{HELD}; \]

On receipt of a request \( <T_i, p_i> \) at \( p_j \) \((i \neq j)\)
if \((state = \text{HELD} \text{ or } (state = \text{WANTED} \text{ and } (T, p_j) < (T_i, p_i))))\)
then
queue request from \( p_i \) without replying;
else
reply immediately to \( p_i \);
end if

To exit the critical section
\[ state := \text{RELEASED}; \]
reply to any queued requests;
Figure 11.5  Multicast synchronization
Maekawa’s algorithm (cont’d on next slide)

On initialization
\[ \text{state} := \text{RELEASED}; \]
\[ \text{voted} := \text{FALSE}; \]

For \( p_i \) to enter the critical section
\[ \text{state} := \text{WANTED}; \]
Multicast request to all processes in \( V_i - \{ p_i \} \);
Wait until (number of replies received = \( K - 1 \));
\[ \text{state} := \text{HELD}; \]

On receipt of a request from \( p_i \) at \( p_j \) (\( i \neq j \))
if \( \text{(state = HELD or voted = TRUE)} \)
then
\[ \text{queue request from } p_i \text{ without replying}; \]
else
\[ \text{send reply to } p_i; \]
\[ \text{voted := TRUE; } \]
end if
Maekawa’s algorithm (cont’d)

For $p_i$ to exit the critical section

\begin{verbatim}
state := RELEASED;
Multicast release to all processes in $V_i - \{p_i\}$;

On receipt of a release from $p_i$ at $p_j$ ($i \neq j$)
if (queue of requests is non-empty)
then
\begin{itemize}
\item remove head of queue – from $p_k$, say;
\item send reply to $p_k$;
\item voted := TRUE;
\end{itemize}
else
\begin{itemize}
\item voted := FALSE;
\end{itemize}
\end{verbatim}

end if
Figure 11.7  A ring-based election in progress

Note: The election was started by process 17. The highest process identifier encountered so far is 24. Participant processes are shown darkened.
Figure 11.8  The Bully Algorithm

Stage 1

Stage 2

Stage 3

Stage 4

The election of coordinator p2, after the failure of p4 and then p3
Figure 11.9  Open and closed groups

Closed group

Open group
Figure 11.10 Reliable multicast algorithm

On initialization
   Received := \{ \};

For process p to R-multicast message m to group g
   B-multicast(g, m); // p ∈ g is included as a destination

On B-deliver(m) at process q with g = group(m)
   if (m ∉ Received)
      then
         Received := Received ∪ \{ m \};
         if (q ≠ p) then B-multicast(g, m); end if
         R-deliver m;
      end if
Figure 11.11  The hold-back queue for arriving multicast messages
Notice the consistent ordering of totally ordered messages $T_1$ and $T_2$, the FIFO-related messages $F_1$ and $F_2$ and the causally related messages $C_1$ and $C_3$ – and the otherwise arbitrary delivery ordering of messages.
Figure 11.13 Display from bulletin board program

<table>
<thead>
<tr>
<th>Item</th>
<th>From</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>A.Hanlon</td>
<td>Mach</td>
</tr>
<tr>
<td>24</td>
<td>G.Joseph</td>
<td>Microkernels</td>
</tr>
<tr>
<td>25</td>
<td>A.Hanlon</td>
<td>Re: Microkernels</td>
</tr>
<tr>
<td>26</td>
<td>T.L’Heureux</td>
<td>RPC performance</td>
</tr>
<tr>
<td>27</td>
<td>M.Walker</td>
<td>Re: Mach</td>
</tr>
<tr>
<td>end</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 11.14 Total ordering using a sequencer

1. Algorithm for group member $p$

   On initialization: $r_g := 0$;

   To TO-multicast message $m$ to group $g$
   \[ B\text{-multicast}(g \cup \{\text{sequencer}(g)\}, <m, i>); \]

   On B-deliver($<m, i>$) with $g = \text{group}(m)$
   \[ \text{Place } <m, i> \text{ in hold-back queue;} \]

   On B-deliver($m_{\text{order}} = <\text{“order”}, i, S>$) with $g = \text{group}(m_{\text{order}})$
   \[ \text{wait until } <m, i> \text{ in hold-back queue and } S = r_g; \]
   \[ \text{TO-deliver } m; \quad // \text{(after deleting it from the hold-back queue)} \]
   \[ r_g = S + 1; \]

2. Algorithm for sequencer of $g$

   On initialization: $s_g := 0$;

   On B-deliver($<m, i>$) with $g = \text{group}(m)$
   \[ B\text{-multicast}(g, <\text{“order”}, i, s_g>); \]
   \[ s_g := s_g + 1; \]
Figure 11.15  The ISIS algorithm for total ordering

Algorithm for group member \( p_i \) \((i = 1, 2, \ldots, N)\)

**On initialization**
\[
V_i^g[j] := 0 \ (j = 1, 2, \ldots, N);
\]

**To CO-multicast message \( m \) to group \( g \)**
\[
V_i^g[i] := V_i^g[i] + 1;
B-multicast(g, \langle V_i^g, m \rangle);
\]

**On B-deliver(\langle V_j^g, m \rangle) from \( p_j \) with \( g = \text{group}(m) \)**
- place \( \langle V_j^g, m \rangle \) in hold-back queue;
- wait until \( V_j^g[j] = V_j^g[j] + 1 \) and \( V_j^g[k] \leq V_i^g[k] \) \((k \neq j)\);
- CO-deliver \( m \);  // after removing it from the hold-back queue
\[
V_i^g[j] := V_i^g[j] + 1;
\]
Figure 11.17  Consensus for three processes

P₁  \(d₁:=\text{proceed}\)  P₂  \(d₂:=\text{proceed}\)

\(v₁=\text{proceed}\)  \(v₂=\text{proceed}\)

\(v₃=\text{abort}\)

P₃ (crashes)
Figure 11.18  Consensus in a synchronous system

Algorithm for process $p_i \in g$; algorithm proceeds in $f + 1$ rounds

On initialization
$\text{Values}_i^1 := \{v_i\}; \text{Values}_i^0 = \{\}$;

In round $r$ ($1 \leq r \leq f + 1$)
$B$-multicast($g, \text{Values}_i^r - \text{Values}_i^{r-1}$); // Send only values that have not been sent
$\text{Values}_i^{r+1} := \text{Values}_i^r$;
while (in round $r$)
{
    On $B$-deliver($V_j$) from some $p_j$
    $\text{Values}_i^{r+1} := \text{Values}_i^{r+1} \cup V_j$;
}

After $(f + 1)$ rounds
Assign $d_i = \text{minimum}($Values$_i^{f+1})$;
Figure 11.19  Three byzantine generals

Faulty processes are shown shaded
Figure 11.20  Four byzantine generals

Faulty processes are shown shaded