Tolerating Crash Failures

Triple Modular Redundancy

Can be generalized to \( N \)-modular redundancy to tolerate up to \( m \) failures using \( N = 2m + 1 \) modules
Tolerating Crash Failures

A reliable channel that tolerates omission failures must guarantee (1) no loss and (2) no duplicate delivery of messages. If the channel is FIFO, then there should be no re-ordering of messages too.

Stenning’s protocol

{program for the sender process S}

define ok : boolean; next : integer;
initially next = 0, ok = true, and both channels are empty;
do ok send (m[next], next); ok:= false
(ack, next) is received ok:= true; next := next + 1
timeout (r,s) send (m[next], next)
od

{program for the receiver process R}

define r : integer; initially r = 0;
do (m[next], s) is received s=r accept the message;
(m[next], s) is received s≠r send (ack, r-1)
od
The sliding window protocol

Creating a **reliable FIFO channel** on top of an unreliable channel that can lose and reorder messages. The requirements are:

- No loss
- No duplication
- No reordering
1. Sender can send up to $w$ messages without receiving acks. If no ack is received, then the entire window of messages is retransmitted.

2. Receiver accepts a message if it is anticipated. Otherwise, it sends back an ack for the last message that it received.

{Sender’s program}

{m[k] = $k^{th}$ message to be transmitted}

\begin{verbatim}
 do  
    last + 1 \leq next \leq last + w \quad send (m[next], next);
          
    next := next + 1

    (ack, j) is received
          \quad if \quad j > last \quad last := j

          \quad j \leq last \quad skip

    fi

    timeout (R, S) \quad next := last + 1 \{Retransmission begins\}
\end{verbatim}
Question 1. Why does it work?

Question 2. Can we solve it using bounded sequence numbers?

Theorem. If the communication channels are non-FIFO, and the message propagation delays are arbitrarily large, then using bounded sequence numbers, it is impossible to design a window protocol that can withstand the loss, duplication, and reordering of messages.
The Alternating Bit Protocol

It is a special version of the window protocol that works only on FIFO channels. The window size $w = 1$.

{Sender's program}

initially next = 0, sent = 1, b = 0;

{Both channels are empty};

do   sent $\neq$ b  $\square$ send (m[next], b);
       next := next +1;
       sent := b

   (ack, j) received $\square$ if j = b $\square$ b := 1-b
                  j $\neq$ b $\square$ skip
   fi

   timeout (r,s) $\square$ send (m[next-1], b)

od
{Receiver’s program}

define \[ j : 0 \text{ or } 1; \]

initially \[ j = 0; \]

\begin{verbatim}
  do (m[next], b) is received \\
    if \[ j = b \] accept the message; \\
       send (ack, j); \\
       j:= 1 – j \\
    \end{verbatim}

\begin{verbatim}
    if \[ j \neq b \] send (ack, 1-j) \\
  fi \\
  od
\end{verbatim}
How TCP works

TCP is a polished version of the sliding window protocol.

*What about generating unique sequence numbers?*

Randomly chosen 32/64-bit pattern is most likely unique. Also, all sequence numbers older than $2^d$ are discarded, where $d$ is the round-trip delay.

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN seq = x</td>
<td>SYN, seq=y, ack = x+1</td>
</tr>
<tr>
<td>SYN, seq=y, ack = x+1</td>
<td>ACK, ack=y+1</td>
</tr>
<tr>
<td>send (m, y+1)</td>
<td>ack (y+2)</td>
</tr>
</tbody>
</table>

3-way handshake