CS 123: CRC Review and Error Recovery

George Varghese

October 16, 2001
Why Error Recovery?

- Transport protocols use same ideas. When we study transport protocols, we will only study differences.
- Many existing protocols like HDLC still use error recovery.
- First non-trivial example of a protocol and problems due to varying message delay and errors (e.g., frame loss and node crashes).
- Hop-by-hop recovery is becoming popular again on links to mobile with high error rates. The wheel of time!
What must Error Recovery Guarantee?

- Packets given for transmission to the sending data link must be delivered to the receiver without duplication, loss, or misordering.
Assumptions

• Layered on top of error detection. Assume undetected error rate is small enough to be ignored.

• Whole frames can be lost in a way that may not be detected by error detection.

• Physical layer is FIFO.

• Delay on links is arbitrary and can vary from frame-to-frame.
Protocol Plays

Node S
\[a, b, c\]

Node R
\[a\]

\[\times\]

\[b\]

Time

Loss

Node S
\[a, b, c\]

Node R
\[a\]

Ret
\[a\]

ack
\[a\]

Duplication
Protocol Plays

Node S

\( a, a, \)  
\( \text{Ret} \)  
\( a \)  
\( \text{ack} \)  
\( a \)  
\( a \)  
\( \text{Livelock} \)

Node R

\( a \)  
\( \text{Reject} \)  
\( a \)  
\( \text{Reject} \)  
\( a \)  
\( \text{Reject} \)

Node S

\( a, b \)  
\( 0 \)  
\( \text{Ret} \)  
\( a, 0 \)  
\( a, 0 \)  
\( a, 0 \)  
\( \text{Livelock} \)

Node R

\( 0 \)  
\( 1 \)  
\( a \)  
\( \text{Reject} \)  
\( \text{Reject} \)  
\( \text{Reject} \)
Protocol Plays

Node S

\[ a, b, c \]

0

Ret \[ a, 0 \]

ack

b, 1

ack

c, 2

Node R

\[ 0 \]

\[ 1 \ a \]

Reject

Loss
Stop and Wait Executions

**Node S**

0

1

2

3

**Node R**

0

1

2

3

- When sender first gets to $n$: no frames with $n$ or acks with $n + 1$ and the receiver is at $n$.

- When receiver first receives frame $n$, entire system only contains number $n$ → only two numbers in system at any time.
Latency and Throughput

- **Throughput**: jobs completed per second. System owners want to maximize this.

- **Latency**: worst-case time to complete a job. Users want to minimize.

- **Propagation Delay**: Time for transmitted bit to reach receiver. Contrast to transmission rate.

- **Pipe Size**: Transmission Rate * Round-trip Propagation Delay. Need to pipeline if pipe size is large. Alternating bit does not.
Sliding Window Protocols

• Sender can send a window of outstanding frames before getting any acks. Lower window $L$, can send up to $L + w - 1$.

• Receiver has a receive sequence number $R$, next number it expects. $L$ and $R$ are initially 0.

• Sender retransmits all frames in current window until it gets an ack. Ack numbered $R$ implicitly acks all numbers $< R$.

• Two variants: receiver accepts frames in order only (go-back-n) or buffers out-of-order frames (selective reject)
GO BACK 3

SELECTIVE REJECT

<table>
<thead>
<tr>
<th>Window Slides</th>
<th>Timer Expires</th>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>D(0) → 1</td>
<td>D(1) → REJECT</td>
<td>D(2) → BUFFER</td>
</tr>
<tr>
<td>D(2) → REJECT</td>
<td>D(3) → Buffer</td>
<td></td>
</tr>
<tr>
<td>0, 2</td>
<td>1, 3</td>
<td>4, 6</td>
</tr>
</tbody>
</table>
Go-back n code

Send (s,m)
The sender can send this frame if:
m corresponds to s-th data item
given to sender by client AND
L <= s <= L + w - 1

Receive(r, Ack)
On receipt:
L := r

Receive(s,m)
On receipt:
If s = R then
R := s + 1
deliver data m to client.

Send(r, Ack)
r must equal R

Selective Reject Code
Send (s,m)
  The sender can send this frame if:
  m corresponds to s-th data item
  given to sender by client AND
  L <= s <= L + w - 1  AND
  s has not been acked.

Receive(r, List, Ack)
  On receipt:
    L = r
    Mark numbers in List as acked

Receive(s,m)
  On receipt:
    If s >= R then
      Mark s as acked; store m
    While R acked do
      Deliver data at position R
      R := R + 1
Send(r, List, Ack)
  r must equal R
  List contains acked numbers > R
Protocol Design Lesson 1

- First design simple protocols; then optimize using what you know of protocol invariants.
- For stop-and-wait, we got away with a space of two numbers \( w + 1 \). Can we do same for sliding window? Depends on sliding window variant used.
Intuition as to why the window size must be bounded

\( w = 2, \text{ Max } = 3 \)

*Case 1*

*Case 2*
Implementation and Other Details

• Timers: works regardless of values, but needed for performance. So calculate round-trip delay.

• Need only one timer (for lowest outstanding number) in Go-back-$n$. Need one for each window element in Sel Reject.

• In selective reject, have to send an ack with $R$ and a bit-map of numbers greater than $R$ that have been received.

• Piggybacking: to reduce frames sent.
FLOW CONTROL

- Windows provide static flow control. Can provide dynamic flow control if receiver acks indicate what receiver will buffer.

Flow control without error recovery

Credits

Rate Control (sender does not send > R frames/sec)
INITIALIZING LINK PROTOCOLS
(in the face of link and node crashes)

Sender
(crash)

Receiver

RESTART

RN=0

ACK

SN=0

D(0)

1

A(1)

EXAMPLE: SETM = RESTART in HDLC
How naive restarts can fail

- Impossibility results tell us what we need to change to get our job done, not just what we can’t do.

- In order to solve crash impossibility: either use non-volatile memory, timers, or random numbers.