Transport Layer III

Dmitri Loguinov
Texas A&M University

March 8, 2022

Original slides copyright © 1996-2004 J.F Kurose and K.W. Ross
Chapter 3: Roadmap

3.1 Transport-layer services
3.2 Multiplexing and demultiplexing
3.3 Connectionless transport: UDP
3.4 Principles of reliable data transfer (cont)
3.5 Connection-oriented transport: TCP
   - Segment structure
   - Reliable data transfer
   - Flow control
   - Connection management
3.6 Principles of congestion control
3.7 TCP congestion control
Pipelined Protocols

- Pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts
  - Range of sequence numbers must be increased
  - Buffering at sender and/or receiver

- Two generic forms of pipelined protocols: Go-Back-N and Selective Repeat
Pipelining: Increased Utilization

First packet bit transmitted, \( t = 0 \)
last bit transmitted, \( t = \frac{L}{R} \)

RTT

ACK arrives, send next packet, \( t = \text{RTT} + \frac{L}{R} \)

First packet bit arrives
last packet bit arrives, send ACK
last bit of 2\textsuperscript{nd} packet arrives, send ACK
last bit of 3\textsuperscript{rd} packet arrives, send ACK

Increases utilization by a factor of 3!

\[ U_{\text{sender}} = \frac{3 \times \frac{L}{R}}{\text{RTT} + \frac{L}{R}} = \frac{0.024}{30.008} = 0.0008 \]
**Go-Back-N (GBN)**

**Sender:**
- **Window** of up to \( N \) consecutive unack’ed pkts allowed
- A field in header that holds \( k \) unique seq numbers

- **ACK(n):** ACKs all consecutive pkts up to \& including seq # \( n \) (cumulative ACK)
  - Means packets 1...\( n \) have been delivered to application

- Timer for the oldest unacknowledged pkt (send_base):
  - Upon timeout: retransmit all pkts in current window (yellow in the figure); reset the timer
GBN: Sender Extended FSM

\[
\begin{align*}
\text{rdt\_send(data)} \quad & \quad \text{if (nextseqnum < base+N)} \{
\text{sndpkt[nextseqnum] = make\_pkt(nextseqnum, data, chksum)} \\
\text{udt\_send (sndpkt[nextseqnum])} \\
\text{if (base == nextseqnum) start\_timer} \\
\text{nextseqnum++}
\} \\
\text{else refuse\_data(data)}
\end{align*}
\]

Wait

\[
\text{timeout} \quad \text{start\_timer} \quad \text{udt\_send(sndpkt[base])} \quad \text{udt\_send(sndpkt[base+1])} \quad \text{...} \quad \text{udt\_send(sndpkt[nextseqnum-1])}
\]

\[
\Lambda \quad \text{base=1} \quad \text{nextseqnum=1}
\]

\[
\text{rdt\_rcv(rcvpkt) && corrupt(rcvpkt)} \quad \text{rdt\_rcv(rcvpkt) && NOT corrupt(rcvpkt)}
\]

\[
\text{new\_base = getacknum(rcvpkt)+1} \quad \text{if (new\_base > base)} \{
\text{base = new\_base;} \\
\text{if (base == nextseqnum)} \\
\text{stop\_timer } \quad // \text{last ACK in window }
\text{else start\_timer }
\}
\]
**GBN: Receiver Extended FSM**

- **ACK-only:** always send ACK for correctly-received pkt with highest *in-order* seq #
  - Duplicate ACKs during loss
  - Need only remember `expectedseqnum`

- **Out-of-order pkt:**
  - Discard $\rightarrow$ no receiver buffering!
  - Re-ACK pkt with highest in-order seq #

\[
\begin{align*}
\text{expectedseqnum} &= 1 \\
\text{sndpkt} &= \text{make_pkt}(0, \text{ACK}, \text{chksum})
\end{align*}
\]

\[
\begin{align*}
\text{udt_send(sndpkt)} \\
\text{rdt_rcv(rcvpkt) \&\& NOT corrupt(rcvpkt) \&\& hasseqnum(rcvpkt,expectedseqnum)} \\
\text{extract(rcvpkt, data)} \\
\text{deliver_data(data)} \\
\text{sndpkt} &= \text{make_pkt(expectedseqnum, ACK, checksum)} \\
\text{udt_send(sndpkt)} \\
\text{expectedseqnum} &= \text{expectedseqnum + 1}
\end{align*}
\]
GBN in Action

Sender (N=4)  Receiver

1
2
3
4  \(\times\)
5
6

ACK1, deliver
ACK2, deliver
ACK2, discard
ACK2, discard
ACK2, discard

timeout
ignore
ignore
ignore

ACK3, deliver
ACK4, deliver
ACK5, deliver
ACK6, deliver
Selective Repeat

- Receiver *individually* acknowledges all correctly received pkts
  - Buffers pkts, as needed, for eventual in-order delivery to upper layer
- Sender only resends pkts for which ACK was not received
  - Separate timer for each unACKed pkt
- Sender window
  - $N$ consecutive packets in [snd_base, snd_base+N-1]
Selective Repeat: Sender, Receiver Windows

Sender (N=7)

Receiver window

Receiver (N=7)

receiver window

received and delivered
received and buffered
expected but not received
available slot

sent & acked
sent & not acked
not sent & available
not available

rcv_base

snd_base

nextseqnum

1 2 3 4 5

rcv_base

1 2 3 4 5
Selective Repeat

**sender**

Data from above:
- If next available seq # in window, send pkt

**Timeout(n):**
- Resend pkt n, restart timer n

**ACK(n) in [snd_base, snd_base+N-1]:**
- Mark pkt n as received
- If n == snd_base, advance snd_base to the next unACKed seq #

**receiver**

Receive pkt n in [rcv_base, rcv_base+N-1]
- Send ACK(n)
- Out-of-order (n>rcv_base): buffer
- In-order (n == rcv_base): deliver, advance rcv_base to next not-yet-received pkt, deliver all buffered, in-order pkts

Pkt n in [rcv_base-N, rcv_base-1]
- ACK(n)

Otherwise:
- Ignore
Selective Repeat in Action (N=4)

timeout on pkt 2

ACK0

ACK1

ACK3

ACK4

ACK5

ACK2
Selective Repeat: Dilemma

Q: How many distinct seq #s are needed for window size N in selective repeat?

Example:

- Seq #’s: 0, 1, 2, 3
- Window size = 3
- Receiver sees no difference in two scenarios!
- Incorrectly passes duplicate data as new in (a)