CMPE 150: Introduction to Computer Networks

Katia Obraczka
Computer Engineering
UCSC Baskin Engineering
Lecture 10
Announcements/Reminders
Announcements/Reminders
Reliable data transfer
Reliable data transfer (rdt) protocols

- **rdt 1.0.**
  - Reliable channel.

- **rdt 2.0**
  - Lossy channel (transmission errors).

- **rdt 2.1**
  - Handles duplicates

- **rdt 2.2**
  - ACK only

- **rdt 3.0**
  - Transmission errors and packet losses.
Mechanisms

- rdt 1.0
- rdt 2.0
  - Error detection with checksum.
  - Feedback.
    - ACKs and NACKs.
- rdt 2.1
  - Sequence numbers.
- rdt 2.2
  - NACK-free.
  - Duplicate ACKs.
- rdt 3.0
  - Retransmission Timer.
rdt3.0 in action

(a) operation with no loss

(b) lost packet
rdt3.0 in action

(c) lost ACK

(d) premature timeout
Why retransmission timers?
Setting up retransmission timers

- Timeout.
- Trade-off:
  - Setting timer too low?
  - Setting timer too high?
Setting up retransmission timers?
Performance of rdt3.0

- rdt3.0 works, but performance “stinks”.
- ex: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

$$d_{trans} = \frac{L}{R} = \frac{8000 \text{bits}}{10^9 \text{bps}} = 8 \text{ microseconds}$$

- $U_{sender}$: utilization - fraction of time sender busy sending

$$U_{sender} = \frac{L / R}{\text{RTT} + L / R} = \frac{0.008}{30.008} = 0.00027$$

- 1KB pkt every 30 msec -> 33kB/sec thruput over 1 Gbps link
- network protocol limits use of physical resources!
**rdt3.0: stop-and-wait operation**

- First packet bit transmitted, $t = 0$
- Last packet bit transmitted, $t = L/R$
- First packet bit arrives
- Last packet bit arrives, send ACK
- ACK arrives, send next packet, $t = RTT + L/R$

The diagram illustrates the timing of packet transmission and acknowledgement in the stop-and-wait protocol.

The formula for the sender's utilisation, $U_{sender}$, is given by:

$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{0.008}{30.008} = 0.00027$$
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, Selective Repeat
ARQ protocols

- Stop-and-wait
- Pipelined protocols
  - Go-back-N
  - Selective repeat
Pipelining: increased utilization

\[ U_{\text{sender}} = \frac{3 \times L / R}{RTT + L / R} \]

First packet bit transmitted, \( t = 0 \)

Last bit transmitted, \( t = L / R \)

First packet bit arrives

Last packet bit arrives, send ACK

Last bit of 2nd packet arrives, send ACK

Last bit of 3rd packet arrives, send ACK

Increase utilization by a factor of 3!
Pipelining Protocols

**Go-back-N: overview**
- **sender**: up to N unACKed pkts in pipeline
- **receiver**: only sends cumulative ACKs
  - doesn’t ACK pkt if there’s a gap
- **sender**: has timer for oldest unACKed pkt
  - if timer expires: retransmit all unACKed packets

**Selective Repeat: overview**
- **sender**: up to N unACKed packets in pipeline
- **receiver**: ACKs individual pkts
- **sender**: maintains timer for each unACKed pkt
  - if timer expires: retransmit only unACKed packet
Go-Back-N

Sender:
- k-bit seq # in pkt header
- “window” of up to N, consecutive unACKed pkts allowed

ACK(n): ACKs all pkts up to, including seq # n - “cumulative ACK”
  - may receive duplicate ACKs (see receiver)
- timer for each in-flight pkt
- timeout(n): retransmit pkt n and all higher seq # pkts in window
GBN in action

sender

send pkt0
send pkt1
send pkt2
send pkt3 (wait)
rcv pkt0
rcv pkt1
rcv pkt2, timeout
send pkt2
send pkt3
send pkt4
send pkt5
rcv pkt4
rcv pkt5
rcv pkt3, discard
rcv pkt2, deliver
send ACK0
send ACK1
send ACK1
send ACK1
send ACK1
rcv pkt3, discard
send ACK2
send ACK3

receiver

r cv pkt0
send ACK0
rcv pkt1
send ACK1
rcv pkt3, discard
send ACK1
rcv pkt4, discard
send ACK1
rcv pkt5, discard
send ACK1
rcv pkt2, deliver
send ACK2
rcv pkt3, deliver
send ACK3

CMPE 150 Winter13
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 not received
  - sender timer for each unACKed pkt
- sender window
  - N consecutive seq #'s
  - again limits seq #s of sent, unACKed pkts
Selective repeat: sender, receiver windows

(a) sender view of sequence numbers

(b) receiver view of sequence numbers
Selective repeat

**sender**

- data from above:
  - if next available seq # in window, send pkt
- timeout(n):
  - resend pkt n, restart timer
- ACK(n) in [sendbase, sendbase+N]:
  - mark pkt n as received
  - if n smallest unACKed pkt, advance window base to next unACKed seq #

**receiver**

- pkt n in [rcvbase, rcvbase+N-1]
  - send ACK(n)
  - out-of-order: buffer
  - in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt
- pkt n in [rcvbase-N, rcvbase-1]
  - ACK(n)
- otherwise:
  - ignore