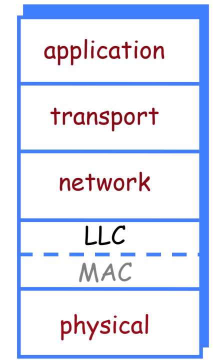
eecs489 COMPUTER NETWORKS

Lecture 26:
Flow Control and ARQ

Data Link Layer

The Data Link layer can be further subdivided into:

1. Logical Link Control (LLC): provides flow and error control
   - different link protocols may provide different services, e.g., Ethernet doesn't provide reliable delivery (error recovery)
2. Media Access Control (MAC): framing and media access



Link Layer Services

Flow Control:
- pacing between adjacent sending and receiving nodes

Error Control:
- errors caused by signal attenuation, noise
- ARQ: receiver detects presence of errors and asks sender for retransmission
- FEC: receiver identifies and corrects bit error(s), without resorting to retransmission

Reliable delivery between adjacent nodes
- seldom used on low bit error links (fiber, some twisted pair)
- plays an important role in wireless links with high error rates
- Q: why do we need both link-level and end-end reliability?

Flow Control

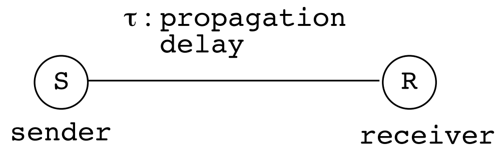
What is flow control?
- receiver telling sender to slow down

Why do you need flow control?

Flow control protocols at data link layer (single hop):
- XON/XOFF
- Stop & Wait Protocol
- Sliding Window Protocol

Similar issues and mechanisms apply at the transport layer (end-to-end)

XON/XOFF



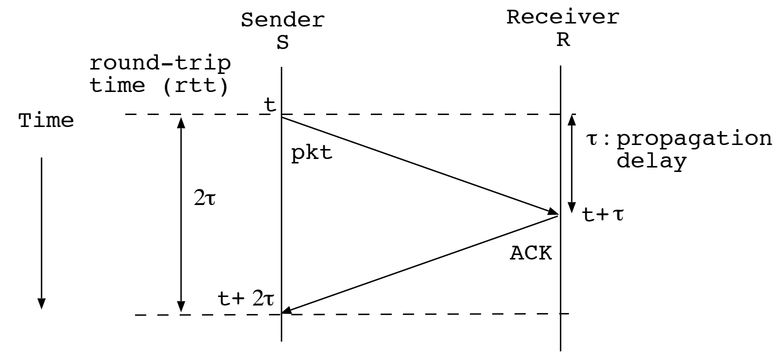
Algorithm:

- S sends stream of data
- R sends XOFF, S stops transmission
- R sends XON, S resumes transmission

Works OK if τ is small, otherwise sender can overrun receiver (Why?)

Stop and Wait (S&W) Protocol

After sending a packet, sender must wait for acknowledgment (ACK) before sending the next packet



Stop & Wait Performance

Disadvantages:

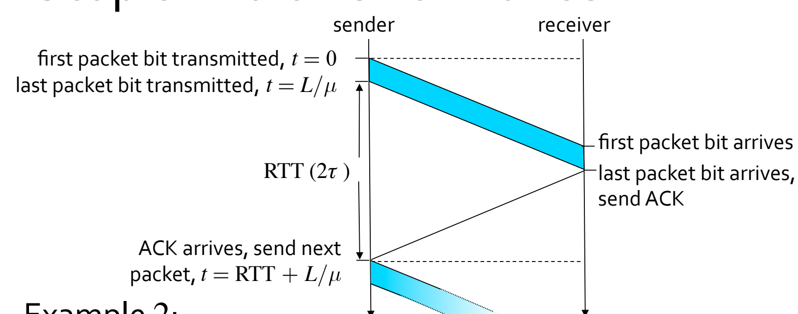
- slow
- must wait for ACK even if no overrun
- max transmission bandwidth 1 packet/round-trip time (rtt)

Performance is ok if τ is small, otherwise inefficient

Example 1:

- link bandwith (μ) = 1 Mbps, with packet size (L) = 1 Kbits, transmission time is $L/\mu = 1$ ms
- if rtt (2τ) = 9 ms, we can send 100 pkts/sec
- the throughput (T_g) is 100 Kbps (10% of capacity)

Stop & Wait Performance



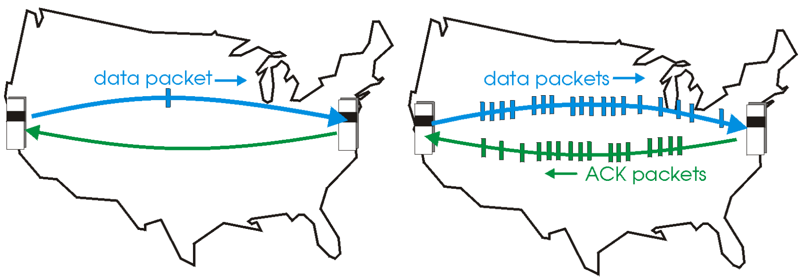
Example 2:

- link bandwidth (μ) = 1 Gbps, propagation delay (τ) = 15 ms, packet size (L) = 8 Kbits, transmission time is $L/\mu = 8\ \mu$s
- sender utilization (U_s), fraction of time sender is sending:

$$U_s = \frac{L/\mu}{2\tau + L/\mu} = \frac{8\cdot 10^3/10^9}{30 + 8\cdot 10^{-6}} = 0.00027$$

Sliding Window: Pipelined Flow Control

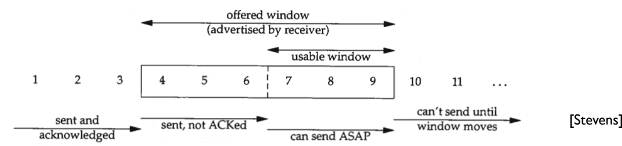
Pipelining: sender allows multiple "in-flight," yet-to-be-acknowledged, packets



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

Sliding Window

Send w number of packets before waiting for an ACK (can have w outstanding, unACKed, packets)



For every received ACK, slide window (over data) by 1 packet (S&W is sliding window with $w = 1$)

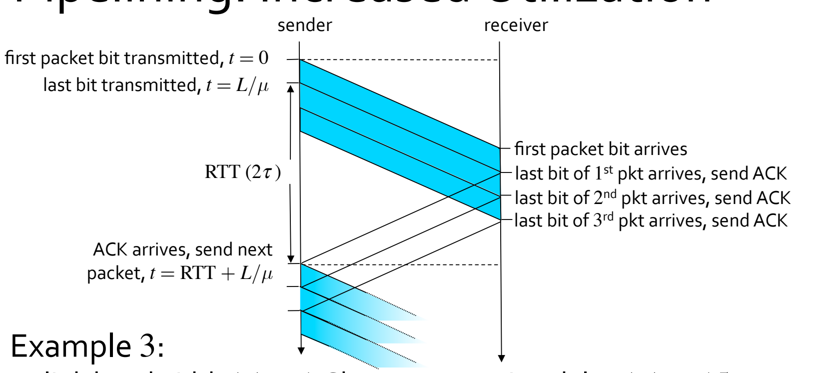
Throughput of the sliding window protocol (T_w):

$$T_w = T_g * w$$

send window size w limited by buffer size at receiver (w_R):

$$T_w = T_g * \mathrm{MIN}(w, w_R)$$

Pipelining: Increased Utilization



Example 3:

- link bandwidth (μ) $= 1$ Gbps, propagation delay (τ) $= 15$ ms, packet size (L) $= 8$ Kbits, transmission time is $L/\mu = 8$ μs, window size (w) $= 3$
- sender utilization (U_s), fraction of time sender is sending:

$$U_s = \frac{w * L / \mu}{2\tau + L/\mu} = \frac{3 * 8 \cdot 10^3 / 10^9}{30 + 8 \cdot 10^{-6}} = 0.0008$$

increased utilization by a factor of 3!

Sliding Window: Max Window Size

What is the optimal window size?
i.e., what's the maximum number of packets on can have outstanding (to "fill the pipe")?

Let μ be the link bandwidth, pipe size $= \mathrm{RTT} * \mu = 2\tau * \mu$ (commonly called the bandwidth-delay product)

Normally you don't want to, and can't, fill the pipe completely (more when we discuss reliable transport protocol)

Error Control

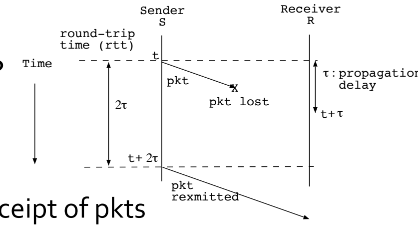
Errors are unavoidable, caused by noise on channel:
- electrical interference, thermal noise, cosmic rays, etc.

Three kinds of transmission errors:
1. sent signal destroyed (data not received)
2. sent signal changed (received wrong data)
3. spurious signal created (received random data)

Automatic Repeat reQuest (ARQ):
sender retransmits lost or corrupted packets

Automatic Repeat reQuest (ARQ)

How does sender know when and which pkts to retransmit?
- by the use of ACKs and timeout

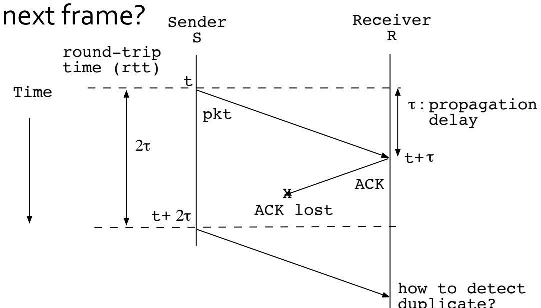


General algorithm:
- receiver acknowledges (ACKs) receipt of pkts
- sender retransmits packets not ACKed by timeout
- a.k.a. PAR: Positive Acknowledgement with Retransmission

Reliability protocols:
- Alternating Bit Protocol (ABP)
- Go-Back-N (GBN, with or without NAK)
- Selective Repeat Protocol (SRP)

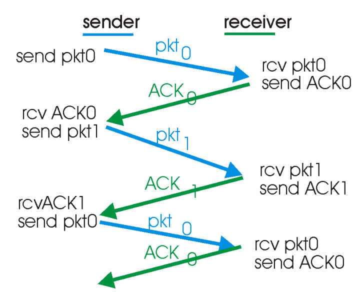
Alternating Bit Protocol (ABP)

S&W with un-numbered packets and ACKs causes confusion on retransmission:
- how to differentiate a retransmitted frame from the next frame?

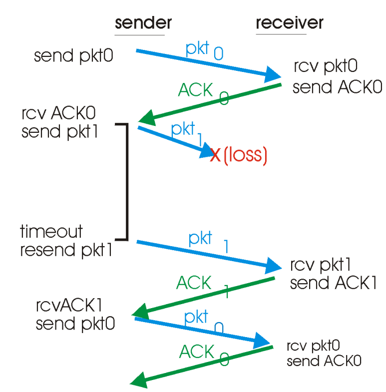


ABP: uses 1 bit to number packets and ACKs

ABP in Action

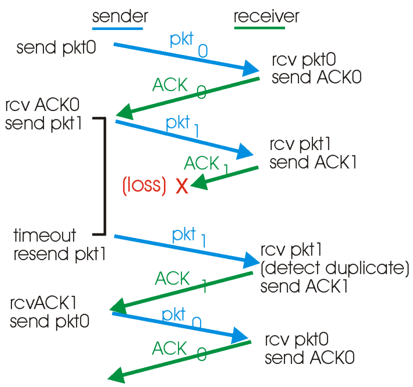


(a) operation with no loss

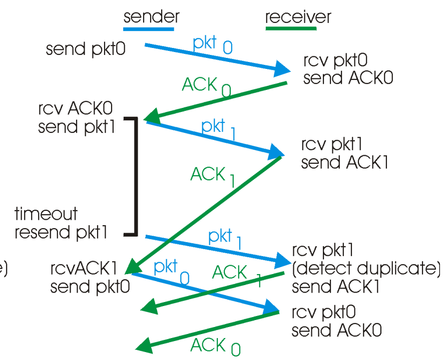


(b) lost packet

ABP in Action



(c) lost ACK



(d) premature timeout

Performance of ABP

ABP works, but performance is bad

Example:

- link bandwidth $(\mu) = 1$ Gbps
- propagation delay $(\tau) = 15$ ms
- packet size $(L) = 8$ Kbits, transmission time is $L/\mu = 8\ \mu$s
- sender utilization (U_s) is the same as S&W:

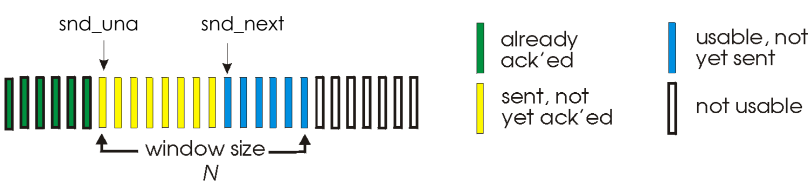
$$U_s = \frac{L/\mu}{2\tau + L/\mu} = \frac{8\cdot 10^3 / 10^9}{30 + 8\cdot 10^{-6}} = 0.00027$$

- about 1 packet every 30 ms
- 33 KBps or 264 Kbps throughput over a 1 Gbps link!

Go-Back-N (Link-layer)

Sender:

- puts k-bit sequence number (seq#) in packet header
- sends a "window" of up to N packets
- consecutive unACKed (`una`) packets allowed

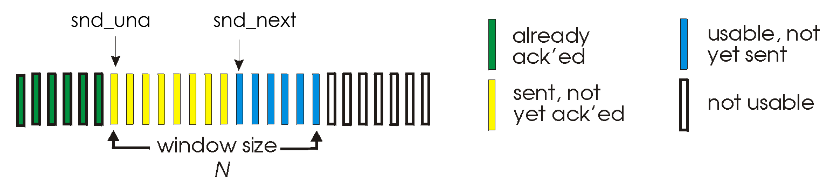


- notation: ACK(n) means ACKs packet with seq# n

Go-Back-N (Link-layer)

Sender:

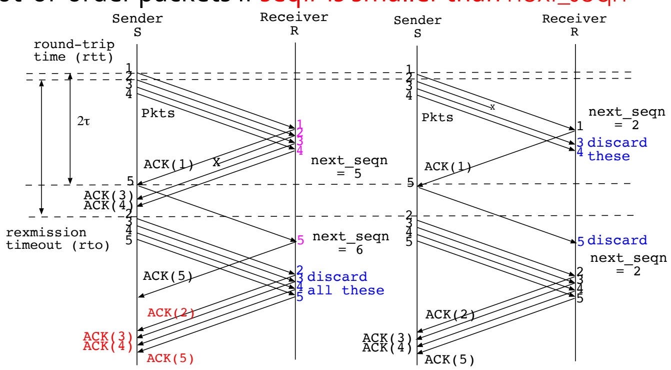
- associates a timer with each in-flight packet
- timeout(n): retransmits packet with seq# n and all higher seq# in window
  - resets snd_next to snd_una (n)
  - resets timer for all retransmitted packets



Go-Back-N (Link-layer)

Receiver:

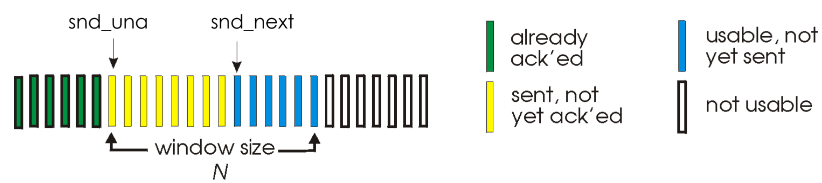
- only needs to remember next expected seq# (next_seqn)
- ACKs and delivers to app packets that arrived in order
- discards out-of-order packets ⇒ no buffering!
- ACKs out-of-order packets if seq# is smaller than next_seqn (why?)



Go-Back-N [K&R]

Sender:

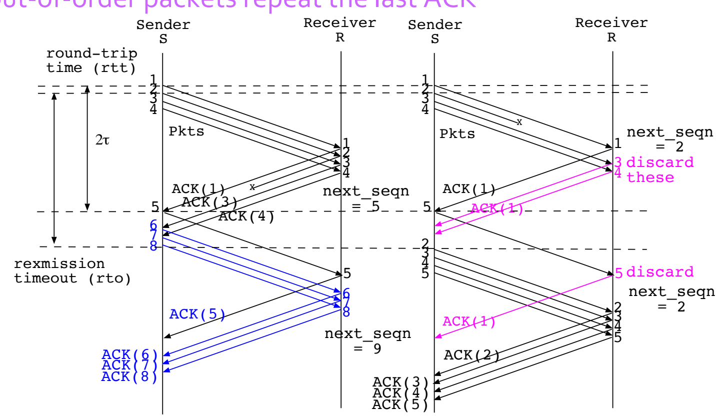
- ACK(n) is cumulative: ACKs all packets up to and including seq# n
- maintains only one active timer, for snd_una
- timeout(snd_una): retransmits snd_una and all higher seq#s in window
  - resets snd_next to snd_una
  - resets timer for snd_una



Go-Back-N [K&R]

Receiver:

- only needs to remember next expected seq# (next_seqn)
- cumulative ACK acknowledges all packets received in-order
- out-of-order packets repeat the last ACK

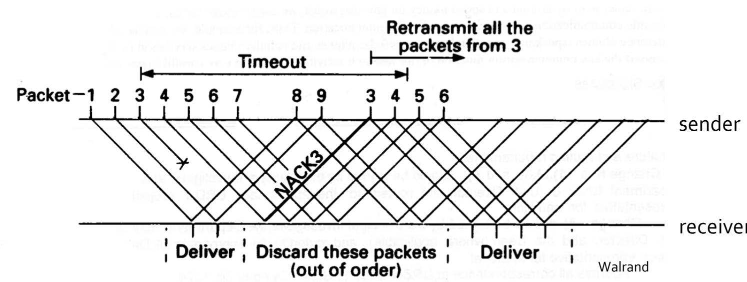


Go-Back-N with Negative ACK (NAK)

Receiver:

- ACKs and delivers in-order packets
- sends NAK for first out of order packet and discards packet
- ACKs and discards subsequent out of order packets

Sender: retransmits on receiving NAK or if RTO expires



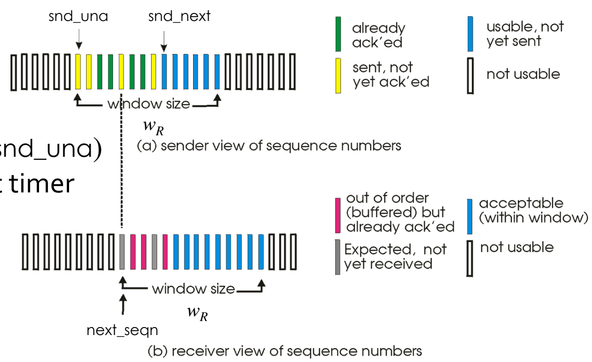
Selective Repeat Protocol (SRP)

Receiver:

- buffers out-of-order packets (up to w_R), for in-order delivery
- ACKs all correctly (no error, but may be out-of-order) received packets individually, not cumulatively

Sender:

- keeps track of w_R and ensures that $w_R >$ (snd_next − snd_una)
- keeps a retransmit timer for each packet
- retransmits only unACKed packets



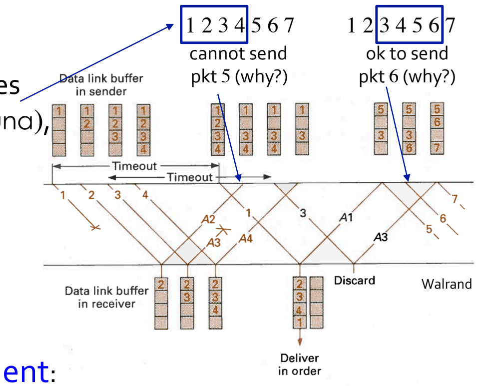
Selective Repeat Protocol (SRP)

Receiver:

- buffers out-of-order packets (up to w_R), for in-order delivery
- ACKs all correctly received packets individually

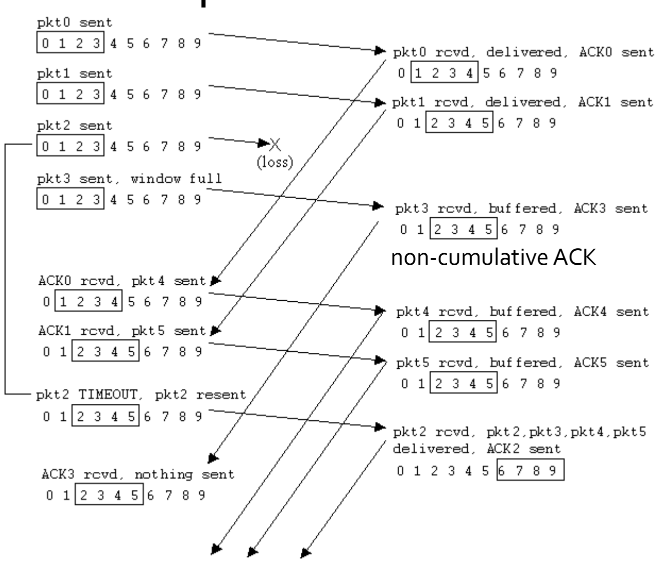
Sender:

- keeps track of w_R and ensures that $w_R >$ (snd_next − snd_una), in the example, $w_R = 4$
- keeps a retransmit timer for each packet
- retransmits only unACKed packets



Selective Acknowledgement:
Piggy-back NAK with ACK., e.g. [ACK2,NAK1], [ACK4,NAK3]

Selective Repeat in Action



Selective Repeat Summary

sender

- data from upper layer:
  - if next available seq# in window, send packet
- timeout(n):
  - resend packet n, restart timer
- received ACK(n) in [snd_una, snd_next]:
  - mark pkt n as received
  - if n is smallest unACKed packet, advance snd_una to next unACKed seq#

receiver

- packet n in [next_seqn, next_seqn+w_R-1]
  - send ACK(n)
  - out-of-order: buffer
  - in-order: deliver (also deliver buffered, in-order packets), advance window to next not-yet-received packet
- packet $n <$ next_seqn
  - send ACK(n)
- otherwise:
  - ignore

ARQ Protocols Summary

Go-Back-N [K&R]:

- sender: allows up to N unACKed packets in pipeline
- receiver: sends cumulative ACKs
  - repeat last ACK if there's a gap
- sender: keeps timer only for oldest unACKed pkt
  - if timer expires: retransmits all unACKed packets

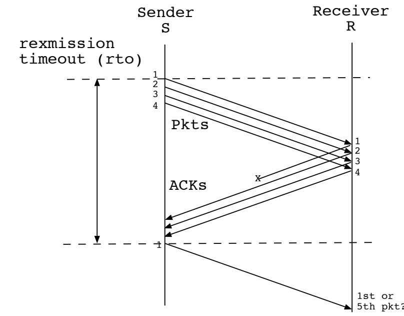
Selective Repeat

- sender: allows up to N unACKed packets in pipeline
- receiver: ACKs individual packets
- sender: maintains timer for each unACKed pkt
  - if timer expires: retransmits only unACKed packet

Simplifying Assumption

Infinite sequence# space

Suppose you have only a 2-bit sequence space:



Q: what's the relationship between seq# space and window size?

Ethernet: Connectionless Service

No handshaking between sending and receiving adaptor

Receiving adaptor doesn't send ACKs or NACKs to sending adaptor

- stream of datagrams passed up to network layer can have gaps
- gaps will be filled if application uses reliable transport layer
- otherwise, application will see the gaps

Other data link protocols may provide error correction and flow control

Other Issues at Transport Layer

Connectionless network layer means each packet can:

- take a different path
- experience different congestion

Implications:

- non-deterministic round-trip time
- out-of-order packets must be buffered for Go-Back-N (so as not to mistake late packets as lost packets)
- complicates computation of receiver's window (w)