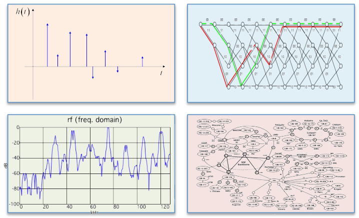


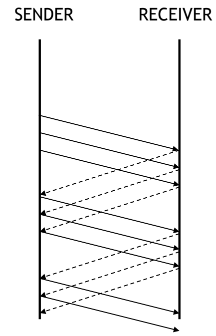
INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Fall 2011
Lecture #23

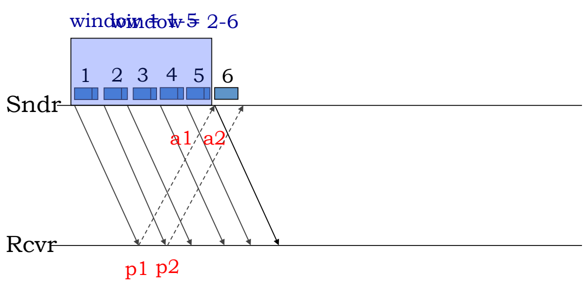
- Sliding window protocol
- Analysis:
  - Bandwidth-delay product & queues
  - Packet loss performance

Sliding Window Protocol



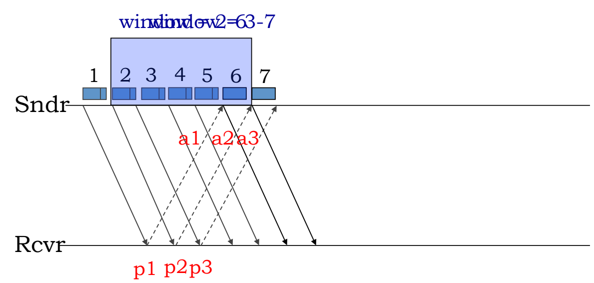
- Use a *window*
  - Allow W packets outstanding (i.e., unack'd) in the network at once (W is called the window size).
  - Overlap transmissions with ACKs
- Sender advances the window by 1 for each in-sequence ack it receives
  - I.e., window *slides*
  - So, idle period reduces
  - **Pipelining**
- Assume that the window size, W, is fixed and known
  - Later, we will discuss how one might set it
  - W = 3 in the example on the left

Sliding Window in Action



W = 5 in this example

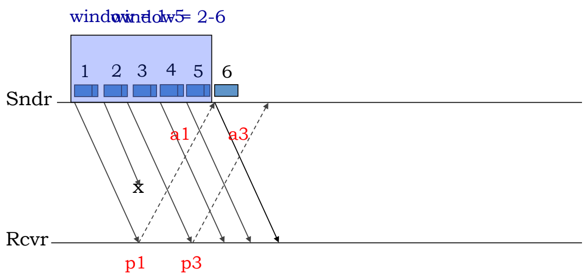
Sliding Window in Action



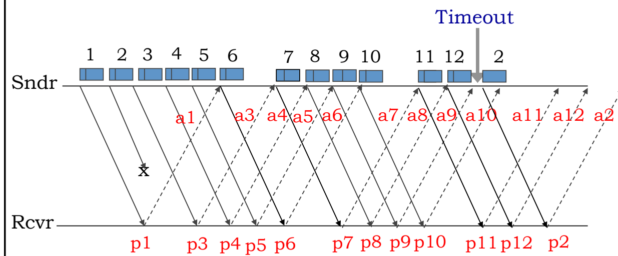
Window definition: If window is W, then max number of *unacknowledged packets* is W

This is a fixed-size sliding window

Sliding Window: Handling Packet Loss

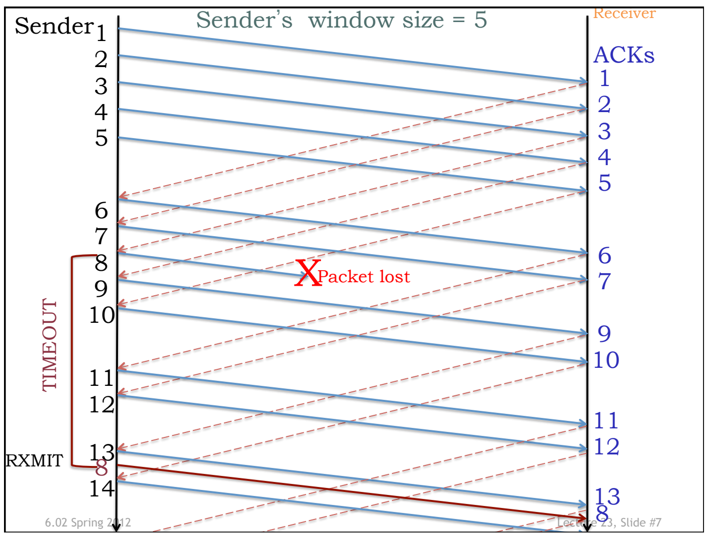


Sliding Window: Handling Packet Loss

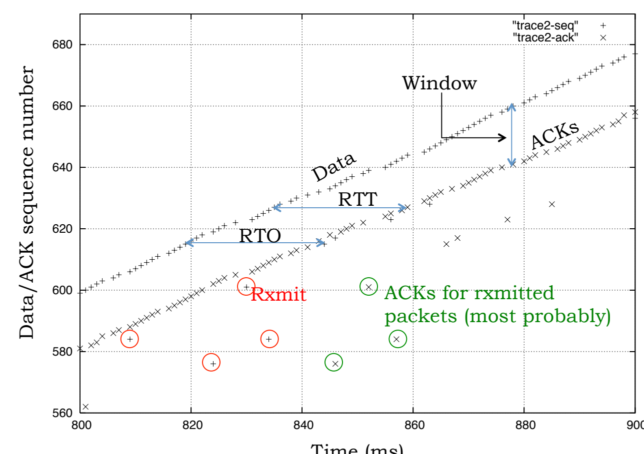


Data packet 2 is lost. The receiver must save packets *all later packets* until packet 2 arrives, to deliver them to the *application* in proper order. Note that with our definition of the window, there's no limit to the number of packets that might arrive out of order.

Q: Can the receiver discard these later packets (3, 4, …, 12?)



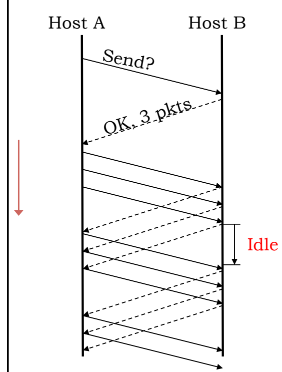
Data/ACK sequence trace



Sliding Window Implementation

- Transmitter
  - Each packet includes a sequentially increasing sequence number
  - When transmitting, save (xmit time,packet) on un-ACKed list
  - Transmit packets if len(un-ACKed list) ≤ window size W
  - When acknowledgement (ACK) is received from the destination for a particular sequence number, remove the corresponding entry from un-ACKed list
  - Periodically check un-ACKed list for packets sent awhile ago
    - Retransmit, update xmit time in case we have to do it again!
    - "awhile ago": xmit time < now − timeout
- Receiver
  - Send ACK for each received packet, reference sequence number
  - Deliver packet payload to application in sequence number order
    - Save delivered packets in sequence number order in local buffer (remove duplicates). Discard incoming packets which have already been delivered (caused by retransmission due to lost ACK).
    - Keep track of next packet application expects. After each reception, deliver as many in-order packets as possible.

Setting the Window Size: Apply Little's Law

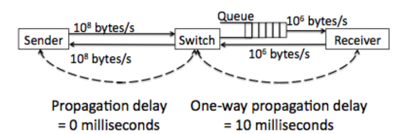


- If we can get "Idle" to 0, will achieve goal
- W = #packets in window
- B = rate of slowest (bottleneck) link
- RTT_min = Min RTT along path, in the absence of any queueing
- If $W = B \cdot$ RTT_min, path will be fully utilized
  - $B \cdot$ RTT_min is the *"bandwidth-delay product"*
  - A *key concept* in the performance of windowed transport protocols

Throughput of Sliding Window Protocol

- If there are no lost packets, protocol delivers W packets every RTT seconds, so throughput is W/RTT
- Goal: to achieve high utilization, select W so that the bottleneck link is never idle due to lack of packets
- Without packet losses:
  - Throughput = W/RTT_{min} if $W \le B \cdot RTT_{min}$,
    = B otherwise
  - If $W > B \cdot RTT_{min}$, then $W = B \cdot RTT_{min} + Q$, where Q is the queue occupancy
- With packet losses:
  - Pick $W > B \cdot RTT_{min}$ to ensure bottleneck link is busy even if there are packet losses
  - *Expected # of transmissions*, T, for successful delivery of pkt and ACK satisfies: $T = (1-L) \cdot 1 + L \cdot (1 + T)$, so $T = 1/(1-L)$, where L = Prob(either packet OR its ACK is lost)
  - Therefore, throughput = $(1-L) \cdot B$
- If $W \gg B \cdot RTT_{min}$, then delays too large, timeout too big, and other connections may suffer

Example

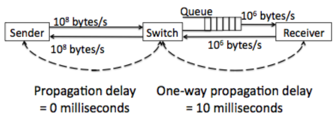


Max queue size = 30 packets
Packet size = 1000 bytes
ACK size = 40 bytes
Initial sender window size = 10 packets

Q: The sender's window size is 10 packets. At what approximate rate (in packets per second) will the protocol deliver a multi-gigabyte file from the sender to the receiver? Assume that there is no other traffic in the network and packets can only be lost because the queues overflow.

A: 10 packets / 21 ms, = 476 packets/second

Example (cont.)



Max queue size = 30 packets
Packet size = 1000 bytes
ACK size = 40 bytes
Initial sender window size = 10 packets

Q: You would like to roughly double the throughput of our sliding window transport protocol. To do so, you can apply one of the following techniques:

a. Double window size W
b. Halve the propagation delay of the links
c. Double the rate of the link between the Switch and Receiver

Q: For each of the following sender window sizes (in packets), list which of the above technique(s), if any, can approximately double the throughput: W=10, W=50, W=30.

Solutions to Example

- Note that BW-delay product on given path = 20 packets
- W=10
  - Doubling window size ~doubles throughput (BW-delay product is 20 on path)
  - Halving RTT ~doubles throughput (since now BW-delay product would be 10, equal to window size)
  - Doubling bottleneck link rate won't change throughput much!
- W=50
  - Doubling window size won't change throughput (we're already saturating the bottleneck link)
  - Halving RTT won't change throughput (same reason)
  - Doubling bottleneck link speed *will* ~double throughput because new bw-delay product doubles to 40, and W=50 > 40
- W=30 (trickiest case)
  - Doubling window size or halving RTT: no effect
  - Doubling bottleneck link changes BW-delay product to 40. W is still lower than 40, so throughput won't double. But it'll certainly increase, by perhaps about 50% more from before