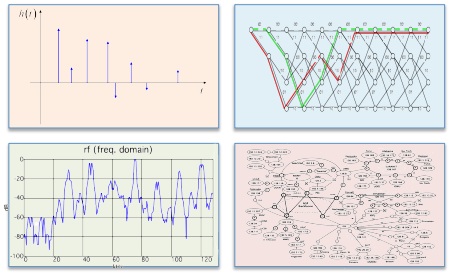


INTRODUCTION TO EECS II

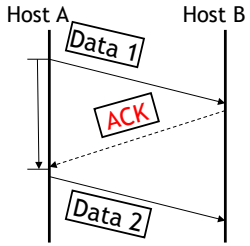
DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2011 Lecture #22

- Sliding-window protocol
- Sizing the window

HKN Eval!

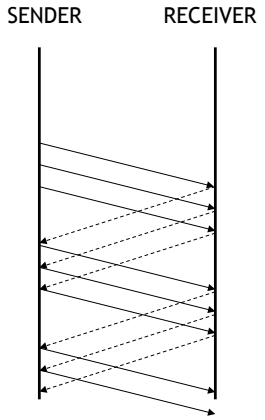
Improving Performance



- Stop-and-wait protocol too slow
- Throughput = 1 packet per RTT
- 1500 byte pkt, 100 ms RTT throughput pegged at 15 KBytes/s
- With packet loss & timeout, throughput even lower

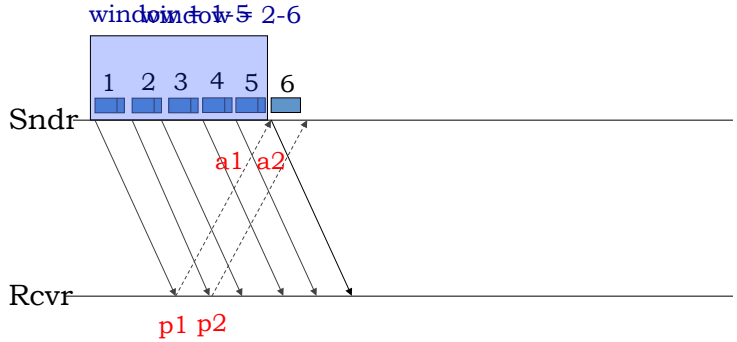
- Solution: Use a *window*
  - Allow W packets outstanding in the network at once (W is called the window size).
  - Overlap transmissions with ACKs

Solution: Use a Sliding Window



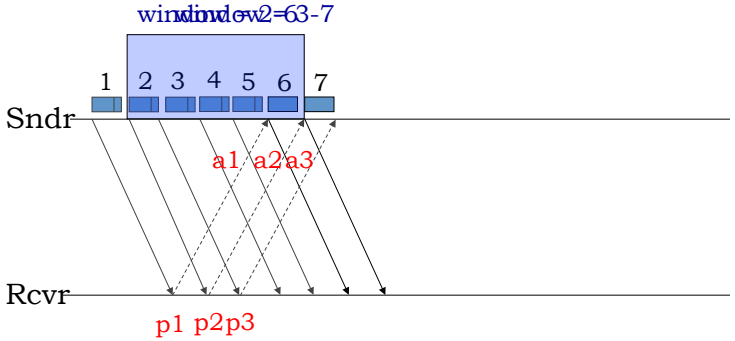
- Senders advances the window by 1 for each in-sequence ack it receives
  - I.e., window *slides*
  - So, idle period reduces
  - **Pipelining**
- Assume that the window size, W, is fixed and known
  - Later, we will discuss how one might set it
  - W = 3 in the example on the left

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

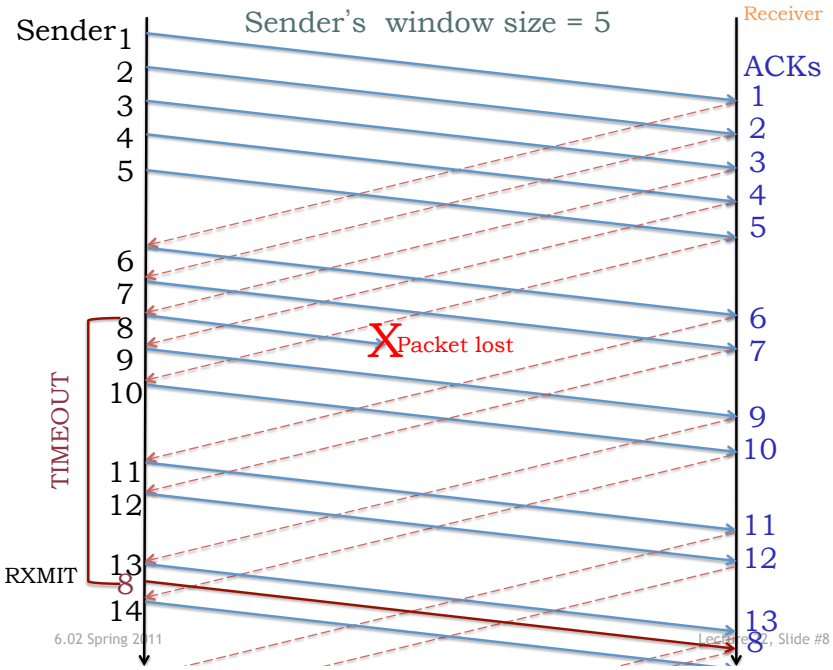
This is a fixed-size sliding window

Sliding Window Implementation

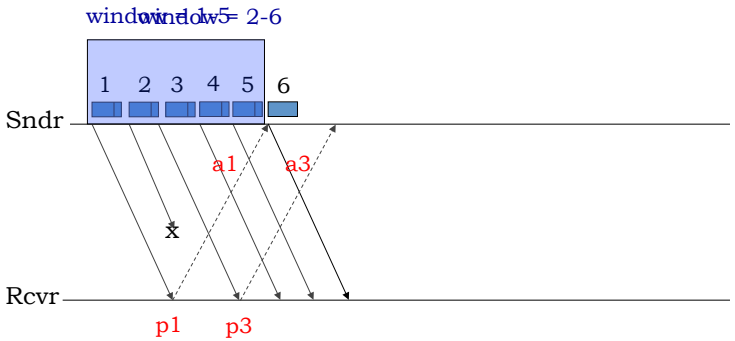
- Transmitter
  - Each packet includes a sequentially increasing sequence number
  - When transmitting, save (xmit time,packet) on un-ACKed list
  - Transmit packets if len(un-ACKed list) ≤ window size W
  - When acknowledgement (ACK) is received from the destination for a particular sequence number, 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.

Issues

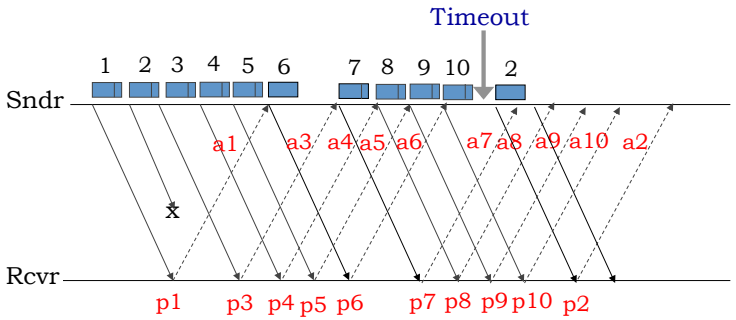
- Timeout chosen as before.
- Size of packet buffers
  - At transmitter
    - Buffer holds un-ACKed packets
    - Stop transmitting if buffer space an issue
  - At receiver
    - Buffer holds packets received out-of-order
    - Stop ACKing if buffer space an issue
- Choosing window size W
  - Too small: some links sit idle, throughput less than maximum
  - Too large: additional packets sit in queues, increasing latency without increasing throughput.
  - Lost packets diminish effective window size (transmitter is limiting itself, but packet is no longer in the network).



Sliding Window: Handling Packet Loss

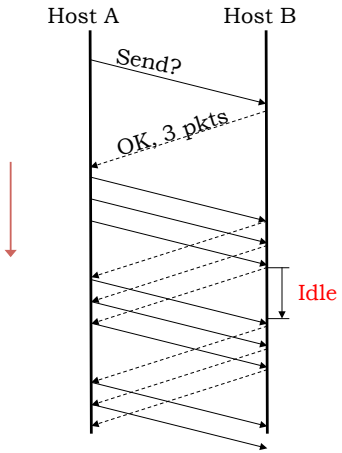


Sliding Window: Handling Packet Loss



The receiver has to save packets 3 through 10 until packet 2 arrives, which will allow it to deliver packets 2 through 10 to the application. Note that with this definition of the window protocol, there's no limit to the number of packets that might arrive out of order.

Setting the Window Size: Apply Little's Law

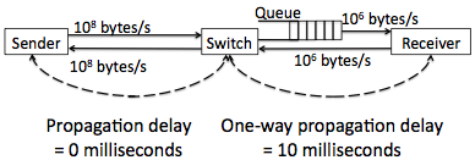


- If we can get "Idle" to 0, will achieve goal
- W = #packets in window
- B = rate of slowest (bottleneck) link
- RTT = avg delay
- If $W = B \cdot RTT$, path will be fully utilized
  - The **"bandwidth-delay product"**
  - Key concept in transport protocols

Throughput of Sliding Window Protocol

- If there are no lost packets, protocol delivers W packets every RTT seconds, so throughput is limited to W/RTT.
  - Maximum throughput is also limited by rate of bottleneck link (B): throughput = min(B, W/RTT)
- Goal: select W so that (slowest) links are never idle due to lack of packets
  - Avoid overfilling queues since that increases packet latency and, if timeouts are triggered, possibility of spurious retransmissions.
    - Measured RTT includes queuing delay = $RTT_{min} + Q_{delay}$
    - As Q_{delay} increases, so does W, which increases Q_{delay}, …
    - Use $B \cdot RTT_{min}$ when calculating W
  - Slightly larger than $B \cdot RTT_{min}$ to ensure bottleneck link is busy even if there are packet losses
    - total # of transmissions, T, for successful delivery
      $T = 1 + L \cdot (1 + L \cdot (1 + \ldots)) = 1 + L + L^2 + \ldots = 1/(1-L)$
      where $L = 1-(1-\text{per\_link\_loss})^{\#\text{ hops in roundtrip}}$ is the round-trip loss rate.
    - Max throughput is $1/T = 1-L = (1-p)^{\#\text{ hops in roundtrip}}$

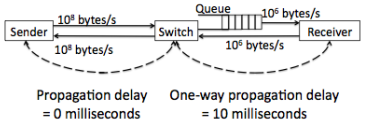
Example



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.

Example

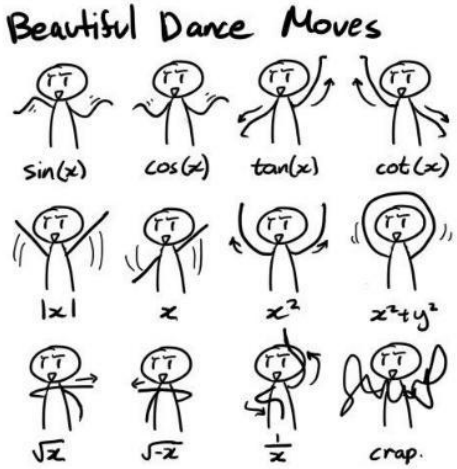


Max queue size = 30 packets
Packet size = 1000 bytes
ACK size = 40 bytes
Initial sender window size = 10 packets

Q: You would like to double the throughput of this sliding window transport protocol. To do so, you can apply <u>one</u> of the following techniques:

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

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



"Shouldn't that be –sin(x)?"

 http://verydemotivational.memebase.com/?s=dancing