

INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

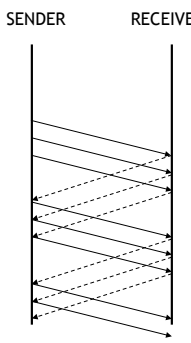
6.02 Fall 2011 Lecture #23

- Redundancy via careful retransmission
- Sequence numbers & acks
- RTT estimation and timeouts
- Stop-and-wait protocol

6.02 Fall 2011
Lecture 23, Slide #1

Sliding Window Protocol

SENDER RECEIVER

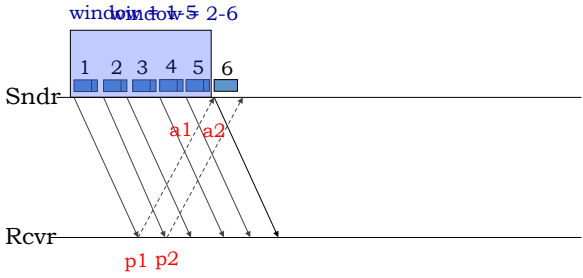


- Use a *window*
 - Allow W packets outstanding (i.e., unack'd) in the network at once (W is called the window size).
 - Overlap transmissions with ACKs
- Sender advances the window by 1 for each in-sequence ack it receives
 - I.e., window *slides*
 - So, idle period reduces
 - **Pipelining**
- Assume 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

6.02 Fall 2011
Lecture 23, Slide #2

Sliding Window in Action

window=5 2-6

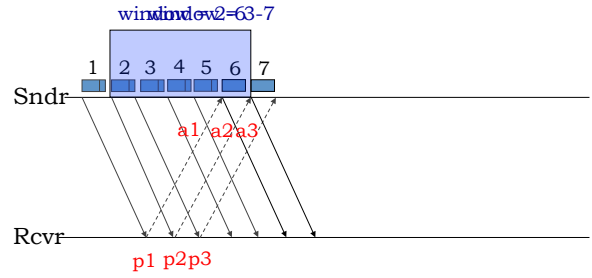


$W = 5$ in this example

6.02 Fall 2011
Lecture 23, Slide #3

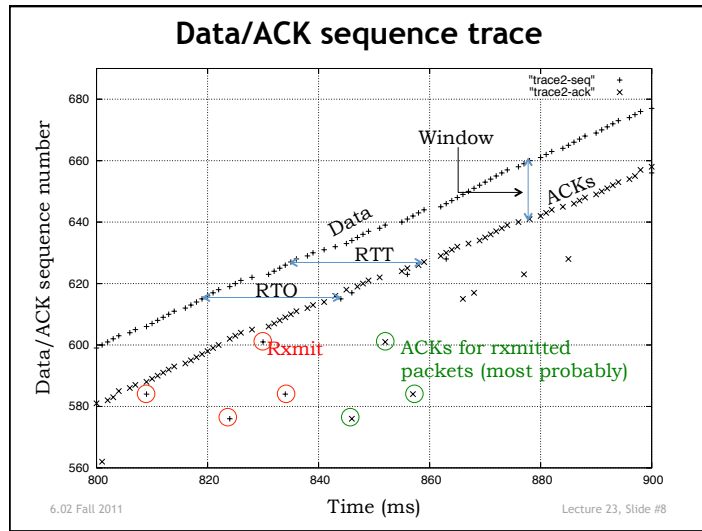
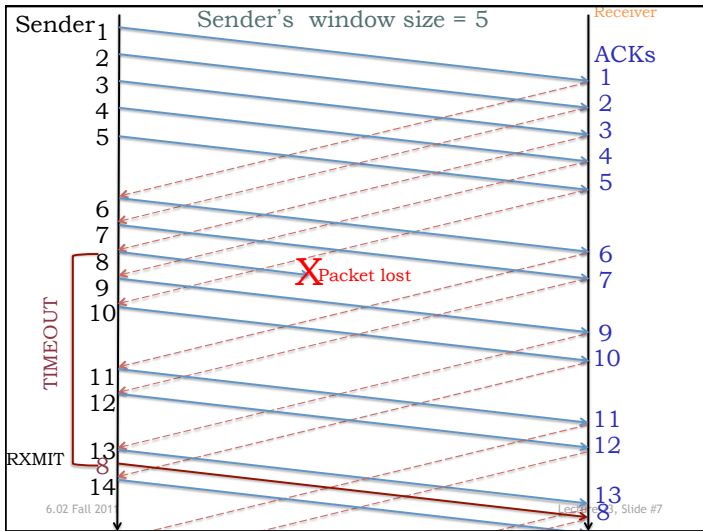
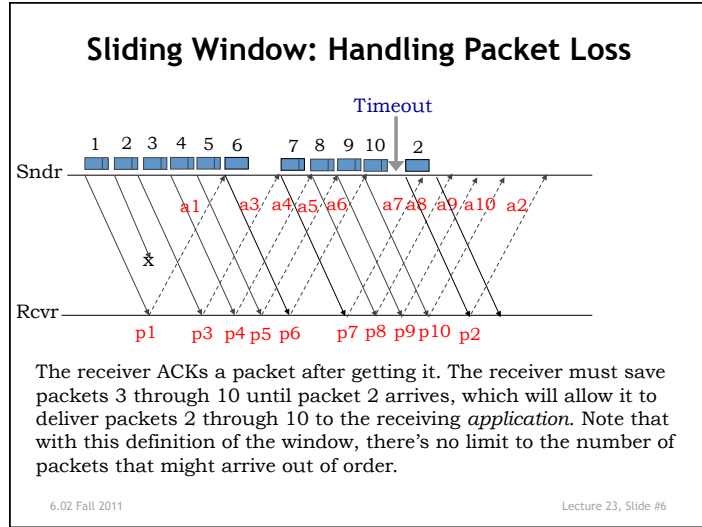
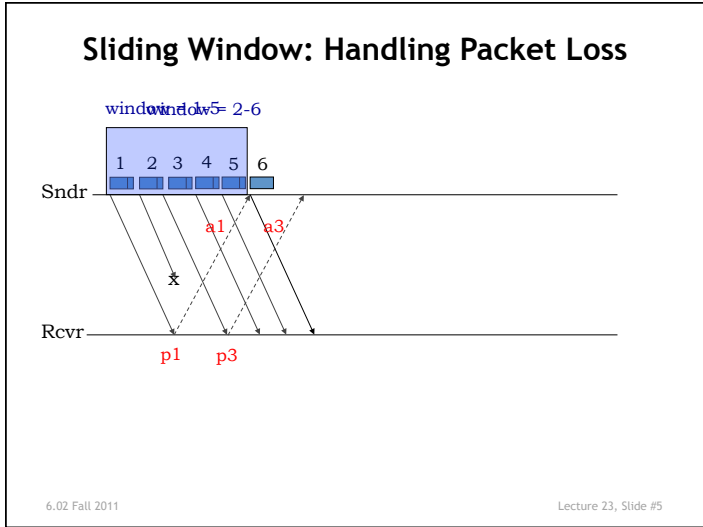
Sliding Window in Action

window=6 3-7



Window definition: If window is W , then max number of unacknowledged packets is W
This is a fixed-size sliding window

6.02 Fall 2011
Lecture 23, Slide #4



Setting the Window Size: Apply Little's Law

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

6.02 Fall 2011 Lecture 23, Slide #9

Throughput of Sliding Window Protocol

- If there are no lost packets, protocol delivers W packets every RTT seconds, so throughput is 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 = $RTT_{\min} + Q_{\text{delay}}$
 - 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 expected # of transmissions, T , for successful delivery
 $T = 1 + L \cdot (1 + L \cdot (1 + \dots)) = 1 + L + L^2 + \dots = 1/(1-L)$
 where $L = 1 - (1 - \text{per_link_loss})^{\text{\#hops_in_roundtrip}}$ is the round-trip loss rate.
 - Throughput is $1/T = 1 - L = (1-p)^{\text{\#_hops_in_roundtrip}}$

6.02 Fall 2011 Lecture 23, Slide #10

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 $\text{len}(\text{un-ACKed list}) \leq \text{window size } W$**
 - 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”: $\text{xmit time} < \text{now} - \text{timeout}$
- 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.**

6.02 Fall 2011 Lecture 23, Slide #11

Example

Propagation delay = 0 milliseconds
 One-way propagation delay = 10 milliseconds

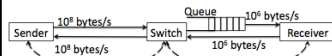
Max queue size = 30 packets
 Packet size = 1000 bytes
 ACK size = 40 bytes
 Initial sender window size = 10 packets

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, ~500 packets/s (actually 476 or so)

6.02 Fall 2011 Lecture 23, Slide #12

Example (cont.)



Propagation delay = 0 milliseconds
 One-way propagation delay = 10 milliseconds

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 this 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 speed 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 speed won't change the throughput!
- $W=50$
 - 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