CSC 4350 – Computer Networks

Lecture 12 – Go-Back-N, Selective Repeat, TCP Intro

October 8, 2024

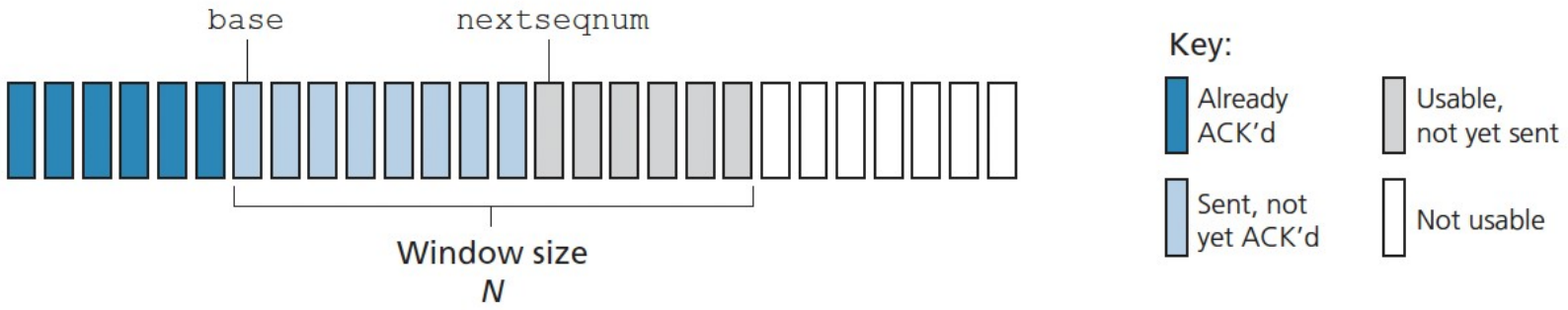
Note

- Material in this lecture is heavily borrowed from Kurose & Ross' "Computer Networking: A Top Down Approach, 8th Edition"

Go-Back-N (GBN)

- Sender is allowed to transmit multiple packets without waiting for acknowledgment
  - Constrained to have no more than some max allowable number N of unacknowledged packets in the pipeline
- Figure 3.19
  - Sender's view of the range of sequence numbers in a GBN protocol
  - Define base to be the sequence number of the oldest acknowledged packet and nextseqnum to be the smallest unused sequence number, then four intervals in the range of sequence numbers can be identified
    - [0, base – 1] – packets that have already been transmitted and acknowledged
    - [base, nextseqnum – 1] – packets that have been sent but not yet acknowledged
    - [nextseqnum, base+N-1] – packets that can be sent immediately, should data arrive from the upper layer
    - Sequence numbers >= base+N cannot be used until an unacknowledged packet currently in the pipeline has been acknowledged

Figure 3.19 – Sender's View of Sequence of Numbers in Go-Back-N



GBN

- Range of permissible sequence numbers for transmitted but not yet acknowledged packets can be viewed as a window of size N over the range of sequence numbers
- As the protocol operates, the window slides forward over the sequence number space
- N is often referred to as window size
- GBN – sliding-window protocol
- Figures 3.20 and 3.21 – FSM description of sender and receiver sides of an ACK-based, NAK-free GBN protocol
- Extended FSM – had variables for base and nextseqnum and added operations on these variables and conditional actions involving those variables
- Starting to look more like a PL specification

GBN Sender Must Respond to Events

- Invocation from above
  - When rdt_send() is called from above, the sender first checks to see if the window is full
    - If not full, a packet is created and sent, variables appropriately updated
    - If full, sender returns the data back to the upper layer, an “implicit indication” that the window is full
      - Upper layer tries again later (probably)
      - Real implementation – sender would probably either have buffered this data or have a synchronization mechanism that would allow for send to be called when window isn’t full
- Receipt of an ACK
  - Acknowledgment for a packet with sequence number n will be taken to a cumulative acknowledgment
    - All packets with a sequence number up to and including n have been correctly received at the receiver

GBN Sender Must Respond to Events

- Timeout event
  - A timer will be used to recover from lost data or acknowledgment of packets
  - If a timeout occurs – the sender resends all packets that have been previously sent but have not been acknowledged
  - Sender in 3.20 – uses a single timer – oldest transmitted but not yet received packet
  - If an ACK is received but there are still additional transmitted but not yet acknowledged packets, the timer is restarted
  - If no outstanding, unacknowledged packets, the timer is stopped

Figure 3.20 – Extended FSM Description of the GBN Sender

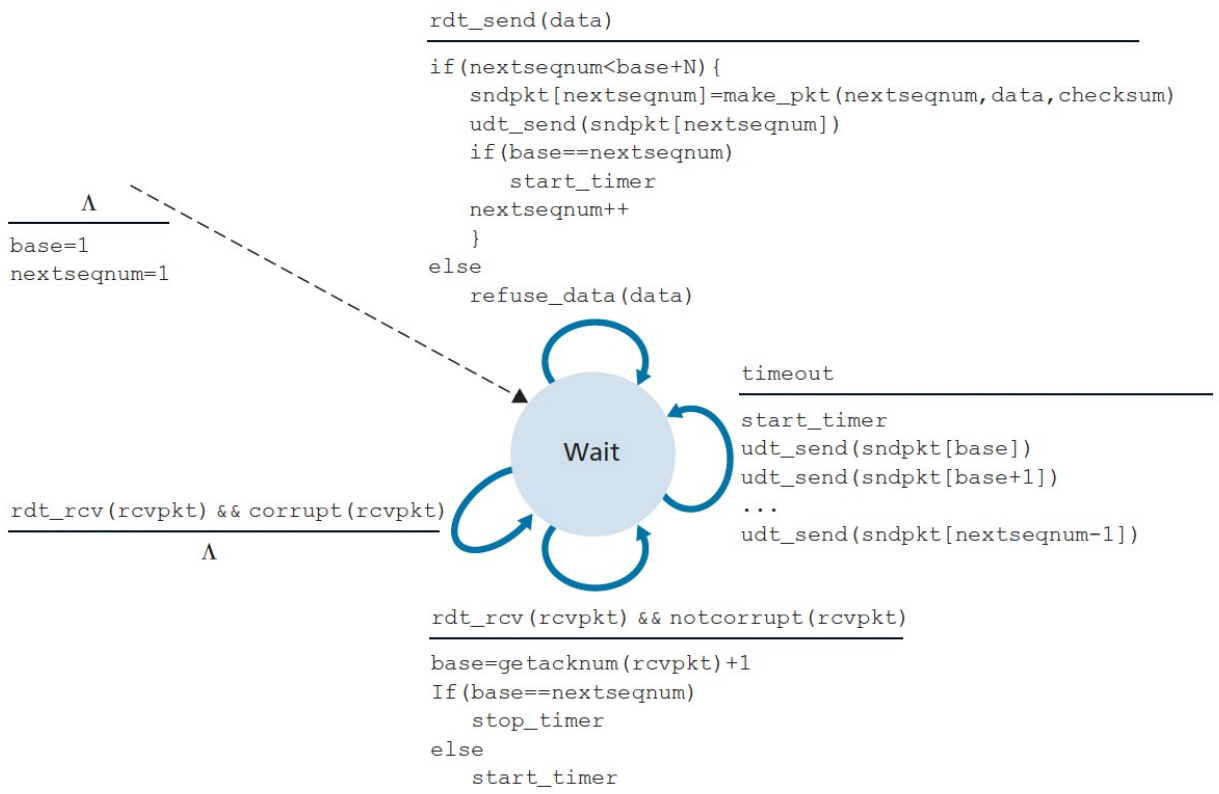
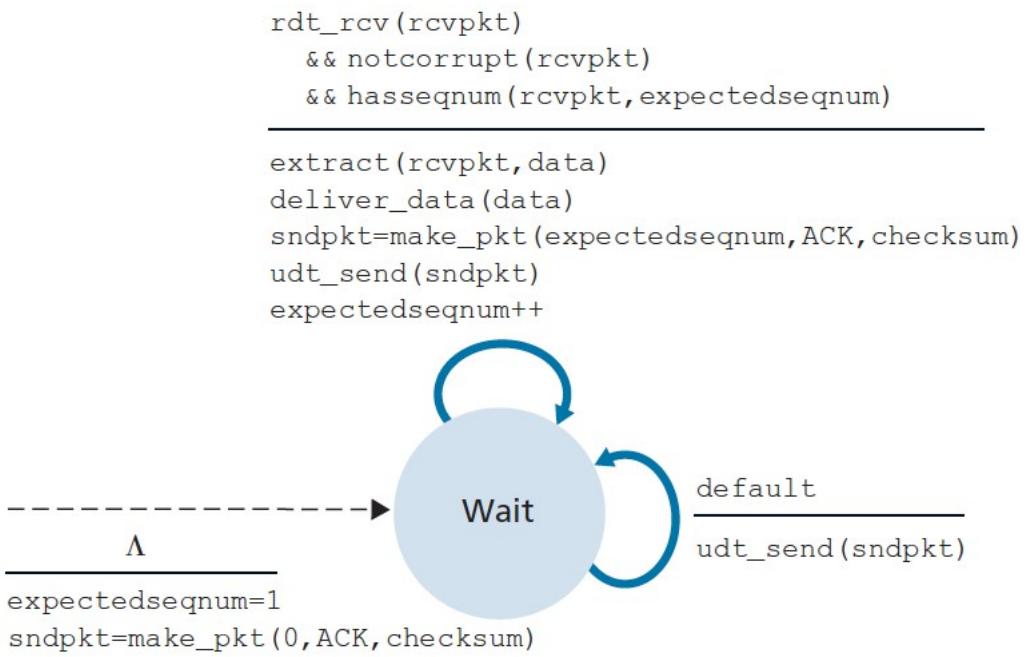


Figure 3.21 – Extended FSM Description of the GBN Receiver



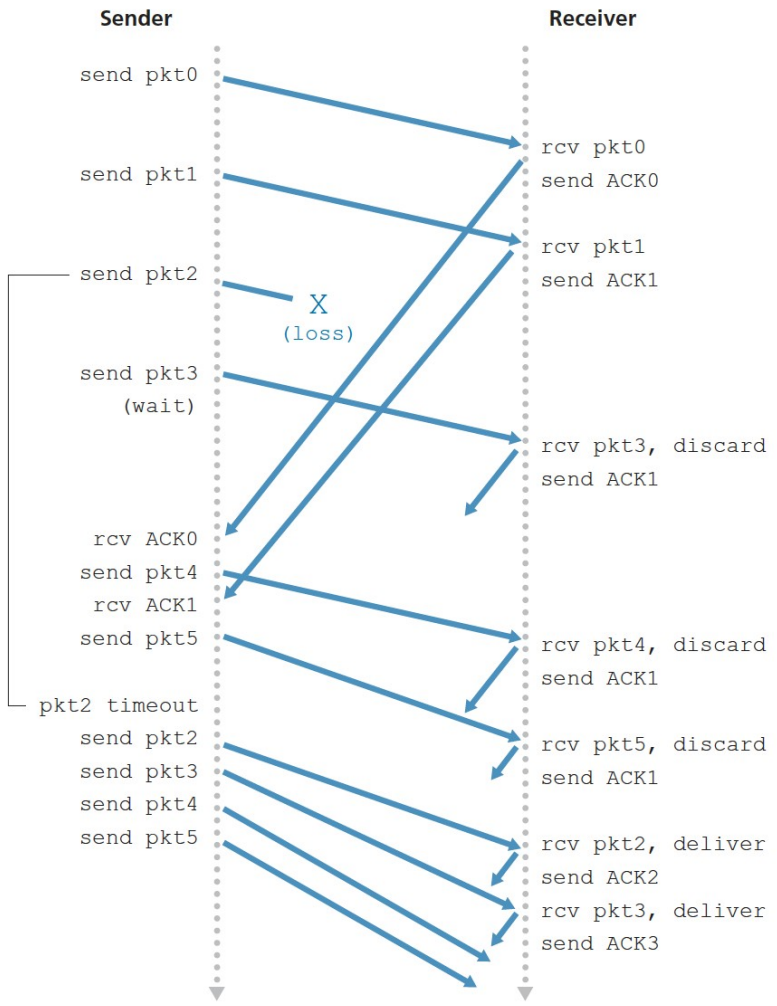
Receiver and GBN

- If a packet with sequence number n is received correctly and is in order, the sender sends an ACK for packet n
  - Delivers the data portion of the packet to the upper level
- Otherwise – receiver discards the packet and resends an ACK for the most recently received in-order packet
- If packet k has been received and delivered, then all packets with a sequence number lower than k have also been delivered
- Receiver discards out-of-order packets
  - Recall: receiver must deliver data in order to the upper layer
  - Suppose n is expected but n+1 arrives – the receiver could buffer and then deliver the packet later after packet n
  - If packet n is lost – both it and packet n+1 will eventually be retransmitted as a result of the GBN retransmission rule
  - Only number receiver needs to maintain is the sequence number of the next in-order packet
    - Value held in the variable expectedseqnum (Fig. 3.21)

GBN

- Figure 3.22 – shows operation of the GBN protocol for the case of a window size of four packets
- Because of the window size limitation, sender sends packets 0-3 but then must wait for one or more of these packets to be acknowledged before proceeding
- As each successive ACK is received, the window slides forward and the sender can transmit one new packet
- Receiver – packet 2 is lost, so 3-5 will be discarded

Figure 3.22 – Go-Back-N Operation



Selective Repeat

- GBN – occasionally can suffer performance setbacks
  - When the window size and the bandwidth-delay product are both large, many packets can be in the pipeline
  - A single packet error can force GBN to retransmit a large number of packets, many of which are not necessary
  - As the probability of channel errors increases, the pipeline can be filled with unnecessary retransmissions
- SR
  - Has the sender retransmit only those packets that it suspects were received in error (lost or corrupted) at the receiver
  - Receiver will need to individually acknowledge correctly received packets
  - Window size of N – limit the number of outstanding, unacknowledged packets in the pipeline
    - But – the sender will have already received ACKs for some of the packets in the window
  - Receiver will acknowledge a correctly-received packet whether or not it is in order
    - Out of order packets are buffered until any missing packets are received
      - Batch of packets can be delivered in order to the upper layer
- Figure 3.24 – various actions taken by the SR sender
- Figure 3.25 – itemizes the various actions taken by the SR receiver
- Figure 3.26 – shows an example of SR operation in the presence of lost packets
  - Receiver initially buffers packets 3-5 and delivers them together with packet 2 to the upper layer once it is received

Figure 3.23 – Selective-Repeat (SR) Sender and Receiver Views of Sequence-Number Space

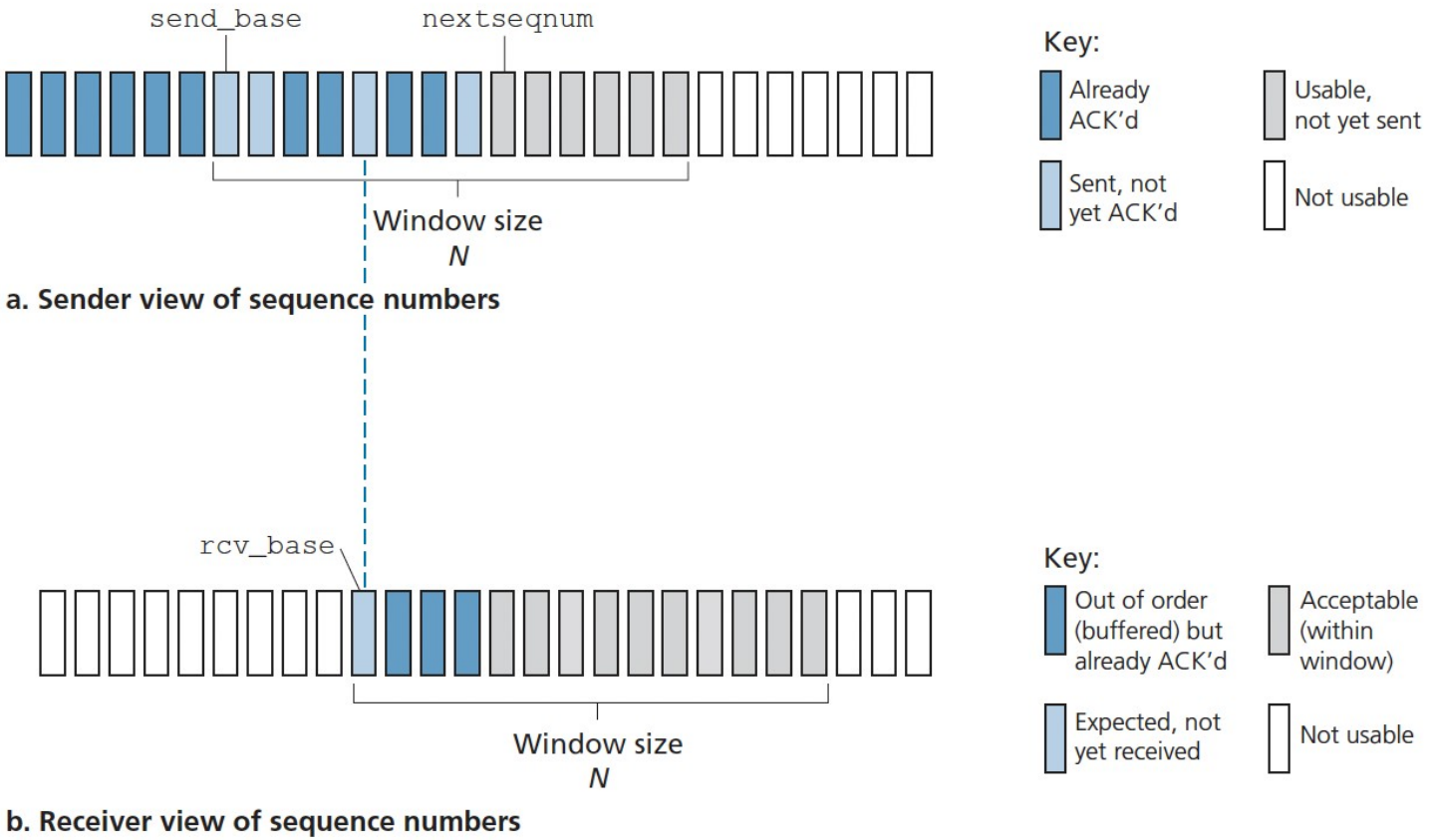


Figure 3.24 – SR Sender Events and Actions

1. *Data received from above.* When data is received from above, the SR sender checks the next available sequence number for the packet. If the sequence number is within the sender's window, the data is packetized and sent; otherwise it is either buffered or returned to the upper layer for later transmission, as in GBN.
2. *Timeout.* Timers are again used to protect against lost packets. However, each packet must now have its own logical timer, since only a single packet will be transmitted on timeout. A single hardware timer can be used to mimic the operation of multiple logical timers [Varghese 1997].
3. *ACK received.* If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window. If the packet's sequence number is equal to `send_base`, the window base is moved forward to the unacknowledged packet with the smallest sequence number. If the window moves and there are untransmitted packets with sequence numbers that now fall within the window, these packets are transmitted.

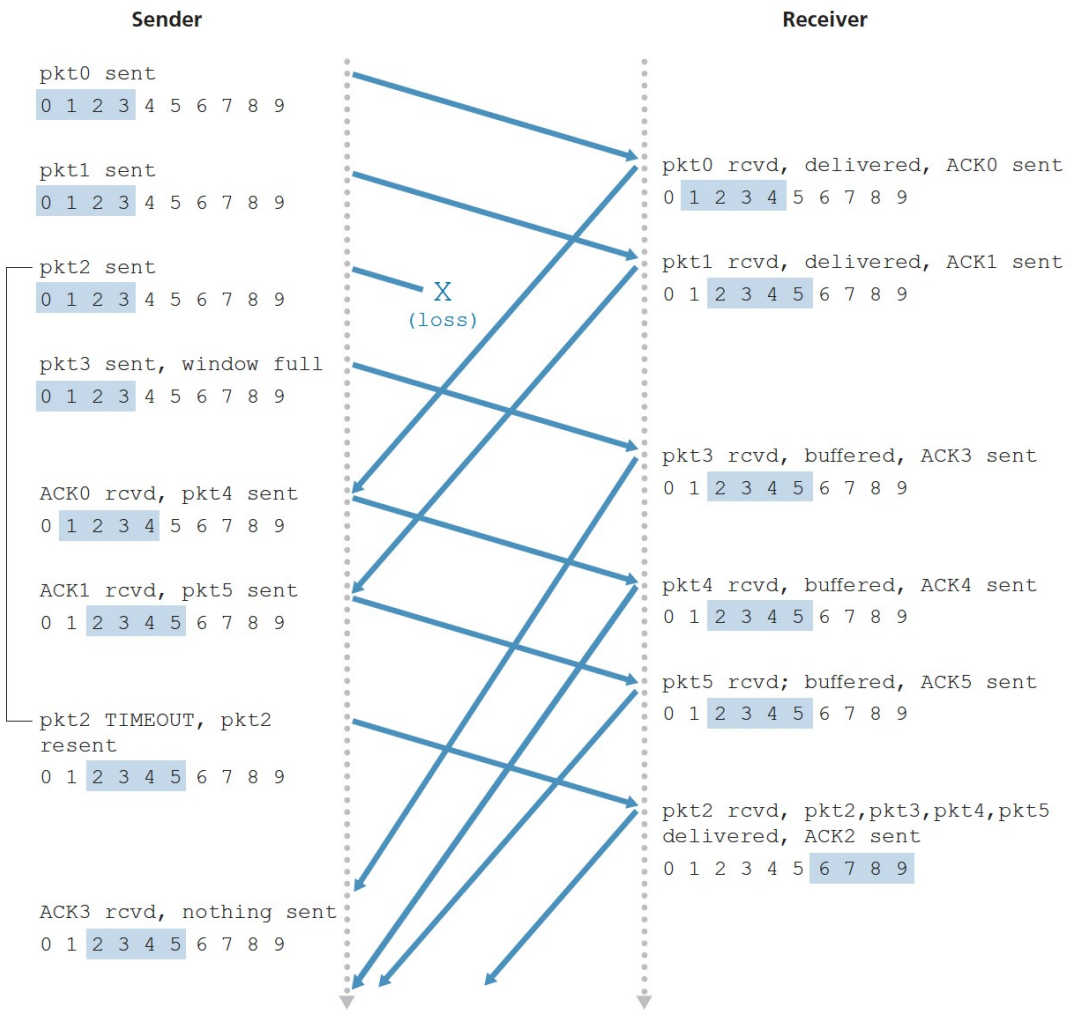
Figure 3.25 – SR Receiver Events and Actions

1. *Packet with sequence number in* `[rcv_base, rcv_base+N-1]` *is correctly received.* In this case, the received packet falls within the receiver's window and a selective ACK packet is returned to the sender. If the packet was not previously received, it is buffered. If this packet has a sequence number equal to the base of the receive window (`rcv_base` in Figure 3.22), then this packet, and any previously buffered and consecutively numbered (beginning with `rcv_base`) packets are delivered to the upper layer. The receive window is then moved forward by the number of packets delivered to the upper layer. As an example, consider Figure 3.26. When a packet with a sequence number of `rcv_base=2` is received, it and packets 3, 4, and 5 can be delivered to the upper layer.
2. *Packet with sequence number in* `[rcv_base-N, rcv_base-1]` *is correctly received.* In this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged.
3. *Otherwise.* Ignore the packet.

More on Figure 3.25

- Receiver reacknowledges already received packets with certain sequence numbers below the current window base
- Given the sender and receiver sequence number spaces in Fig. 3.23, if there is no ACK for packet send_base propagating from receiver to sender, the sender will eventually retransmit the packet send_base
- If the receiver does not acknowledge this packet, the sender's window would never move forward
- Important aspect of SR: sender and receiver will not always have an identical view of what has been received correctly and what has not
  - Sender and receiver windows will not always coincide

Figure 3.26 – SR Operation



SR

- One issue – we have a finite range of sequence numbers
  - Example – with a finite range of four packet sequence numbers 0, 1, 2, 3 and a window size of 3
  - Packets 0 through 2 are transmitted/correctly received and acknowledged
  - Receiver's window is now over the fourth, fifth, and sixth packets, which have sequence numbers 3, 0, and 1
  - Scenarios
    - Figure 3.27a – ACKs for the first three packets are lost and the sender retransmits these packets; receiver will then receive a packet with sequence number 0 - a copy of the first packet sent
    - Figure 3.27b – the ACKs for the first three packets are all delivered correctly. Sender moves forward and sends the fourth, fifth, and sixth packets, with sequence numbers 3, 0, and 1
      - Packet with sequence number 3 is lost, but packet with sequence number 0 – packet containing new data

SR

- Another issue/point of view – the receiver
  - As far as Figure 3.27 is concerned, both of the scenarios are identical
  - No way to distinguish retransmission of the first packet from an original transmission of the fifth packet

Figure 3.27a – SR Receiver Dilemma with Too-Large Windows – A New Packet?

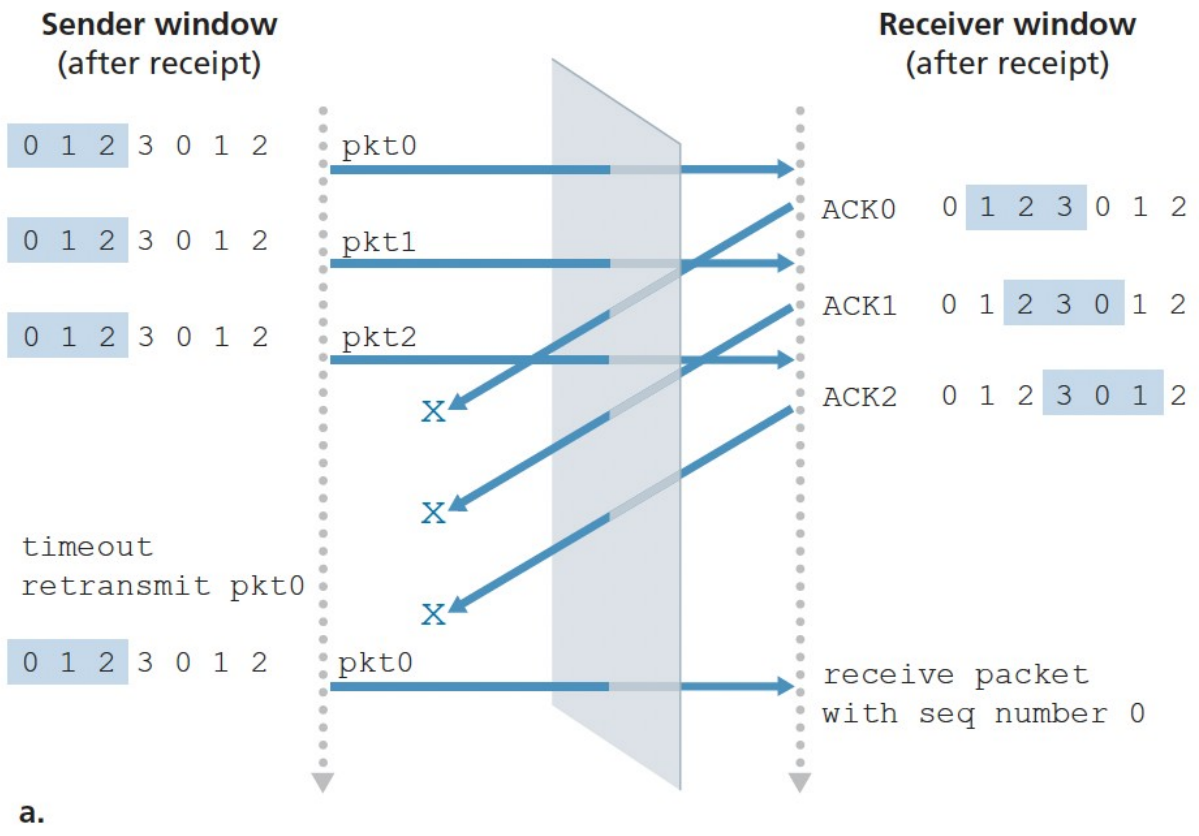


Figure 3.27b - SR Receiver Dilemma with Too-Large Windows – Retransmission?

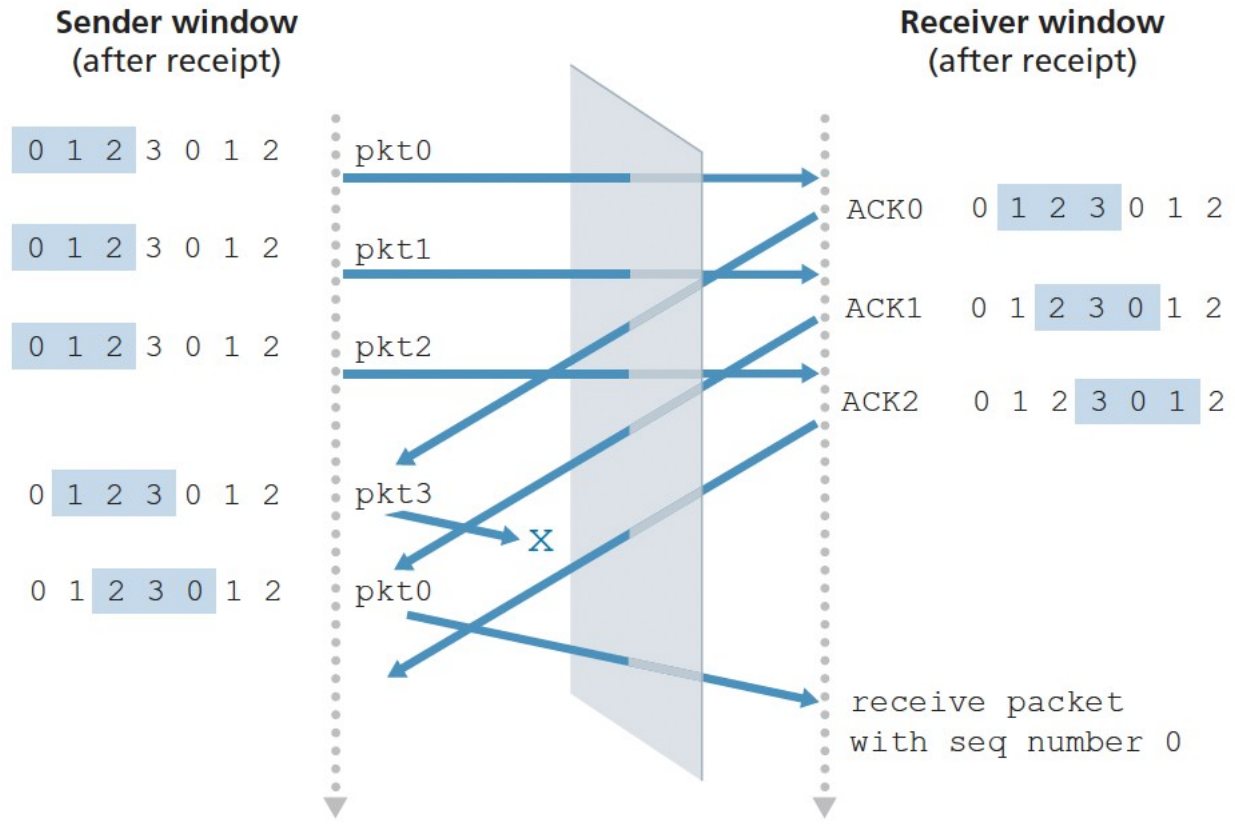


Table 3.1 – Summary of Reliable Data Transfer Mechanisms and Their Use

Mechanism	Use, Comments
Checksum	Used to detect bit errors in a transmitted packet.
Timer	Used to timeout/retransmit a packet, possibly because the packet (or its ACK) was lost within the channel. Because timeouts can occur when a packet is delayed but not lost (premature timeout), or when a packet has been received by the receiver but the receiver-to-sender ACK has been lost, duplicate copies of a packet may be received by a receiver.
Sequence number	Used for sequential numbering of packets of data flowing from sender to receiver. Gaps in the sequence numbers of received packets allow the receiver to detect a lost packet. Packets with duplicate sequence numbers allow the receiver to detect duplicate copies of a packet.
Acknowledgment	Used by the receiver to tell the sender that a packet or set of packets has been received correctly. Acknowledgments will typically carry the sequence number of the packet or packets being acknowledged. Acknowledgments may be individual or cumulative, depending on the protocol.
Negative acknowledgment	Used by the receiver to tell the sender that a packet has not been received correctly. Negative acknowledgments will typically carry the sequence number of the packet that was not received correctly.
Window, pipelining	The sender may be restricted to sending only packets with sequence numbers that fall within a given range. By allowing multiple packets to be transmitted but not yet acknowledged, sender utilization can be increased over a stop-and-wait mode of operation. We'll see shortly that the window size may be set on the basis of the receiver's ability to receive and buffer messages, or the level of congestion in the network, or both.

Connection-Oriented Transport – TCP

- The TCP Connection
- TCP Segment Structure
  - Telnet – A Case Study for Sequence and Acknowledgment Numbers
- Round-Trip Time Estimation and Timeout
  - Estimating the Round-Trip Time
- Reliable Data Transfer
- Flow Control
- TCP Connection Management

The TCP Connection

- TCP is considered connection-oriented because before one application process can begin to send data to another, the two processes must first “handshake”
  - Both sides of the connection will initialize many TCP state variables associated with the TCP connection
- The connection is a logical one, with common state residing only in the TCPs in the two communicating end systems
- Recall: intermediate network elements do not maintain TCP connection state
  - Routers see datagrams, not connections
- Provides full-duplex service
  - Application-layer data can flow from Process 1 to Process 2 at the same time an application-layer data flows from Process 2 to Process 1
- TCP connection is always point-to-point – between a single sender and single receiver

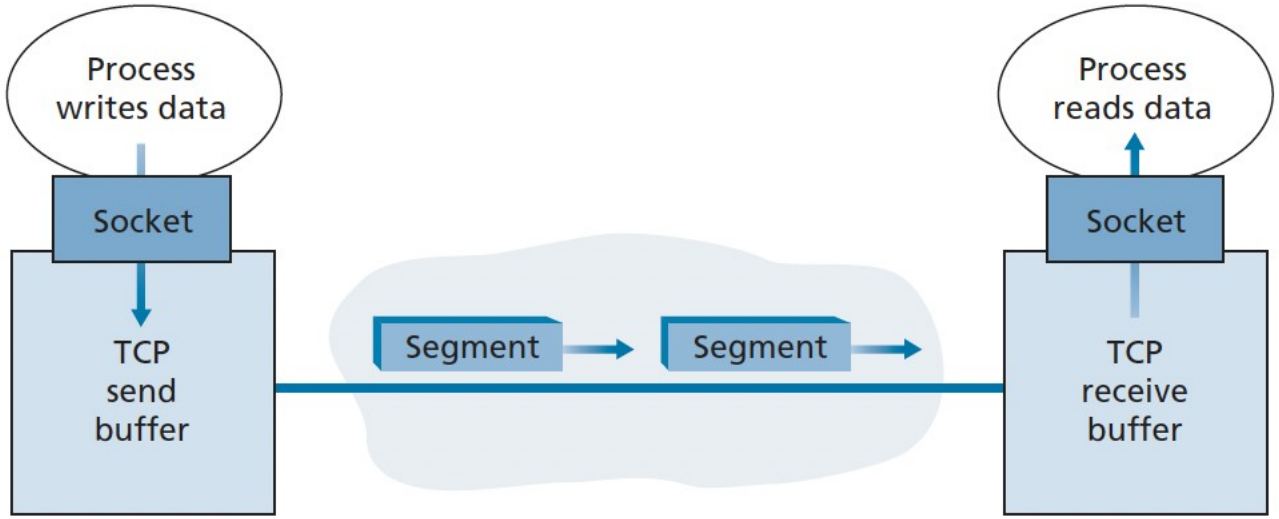
Establishing the Connection

- Process running in one host wants to initiate a connection with another process in another host
  - Client process – initiating the connection
  - The other process – server process
- Client application process first informs the client transport layer that it wants to establish a connection to a process in the server
  - Issued by a command: clientSocket.connect((serverName, serverPort))
    - Python example
    - serverName – name of the server
    - serverPort – process on the server
  - TCP then attempts to establish a TCP connection with TCP in the server
  - Three-way handshake
    - Client – sends an initial message
    - Server – responds with its own initial message to client
    - Client – sends the next segment; may/may not have a payload
- Once established, the application processes can send data to each other

Establishing the Connection, Cont.

- Client process passes a stream of data through the socket (the door of the process)
- Once data passes through the door, the data is in the hands of the TCP running in the client
  - TCP directs this data to the connection's send buffer – one of the buffers that is set aside during the initial handshake
    - Figure 3.28
  - TCP will occasionally grab chunks of data from the send buffer and pass the data to the network layer
  - TCP spec is "laid back" about specifying when TCP should actually send buffered data
  - Maximum Segment Size – max amount of data that can be grabbed and placed in a segment
  - MSS – typically set by determining the length of the largest link-layer frame that can be sent by the local sending host (max transmission unit, MTU)
    - Set the MSS to make sure that a TCP segment will fit into a single link-layer frame
    - Both Ethernet and PPP link-layer protocols have an MTU of 1500 bytes, so a typical value for MSS is 1460 bytes

Figure 3.28 – TCP Send and Receive Buffers



Establishing the Connection, Continued

- TCP pairs each chunk of client data with a TCP header, forming TCP segments
- Segments are passed down to the network layer
  - Separately encapsulated within network-layer IP datagrams
  - Datagrams are then sent into the network
- When TCP receives a segment at the other end, the segment's data is placed in the TCP connection's receive buffer
- Application reads the stream of data from this buffer
- Each side of the connection has its own send and receive buffers