CSC 4350 – Computer Networks

Discussing TCP

October 10, 2024

Note

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

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

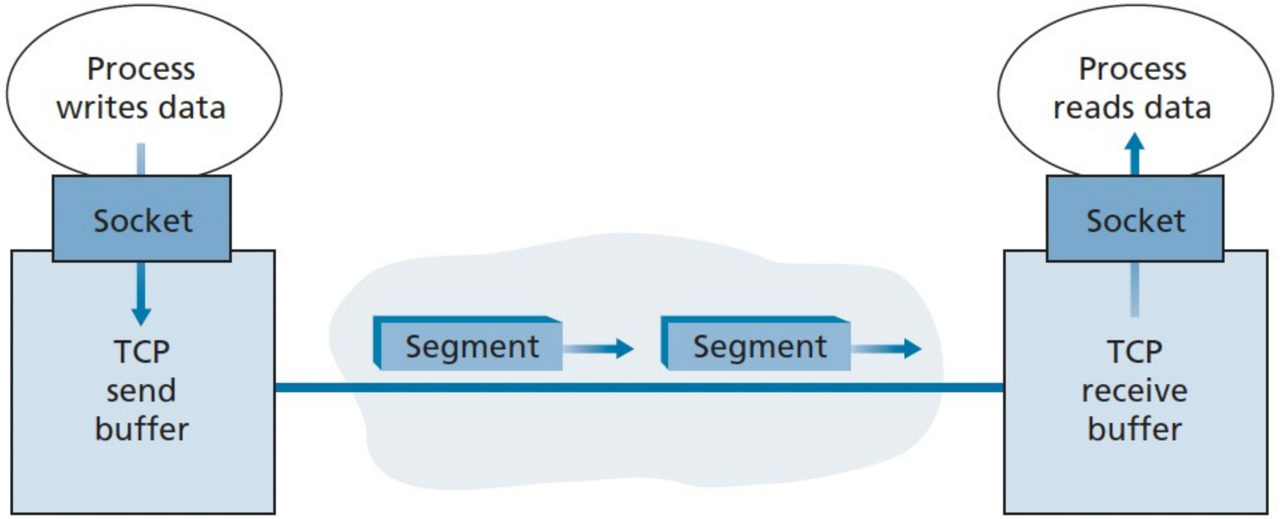
Establishing the Connection

- 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



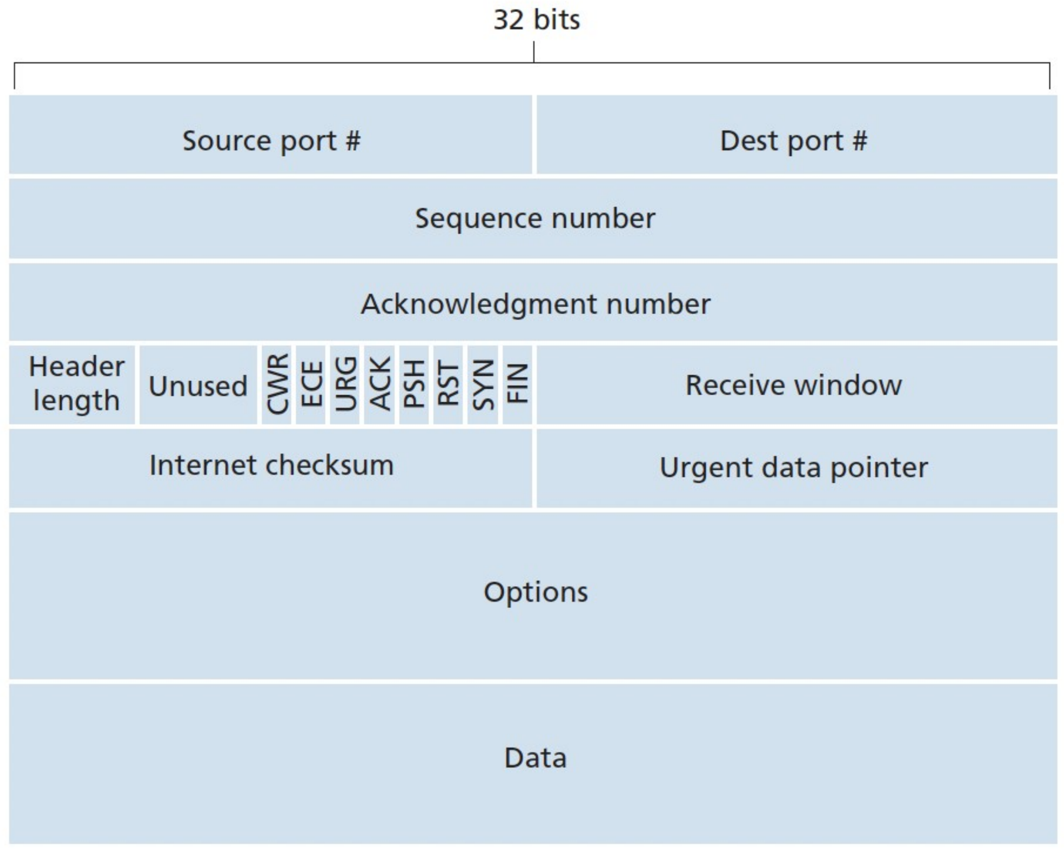
Establishing the Connection, Continued

- 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

TCP Segment Structure

- 32-bit sequence number, 32-bit acknowledgment number – used by TCP sender and receiver in implementing reliable data transfer service
- 16-bit receive window field – flow control; used to indicate the number of bytes that a receiver is willing to accept
- 4-bit header length field – length of the TCP header in 32-bit words
    - TCP header can be of variable length due to TCP options field
- Optional and variable-length options field – when a sender and receiver negotiation the max segment size or as a window scaling factor for use in high-speed networks
- Flag field – 6 bits – used to indicate that the value carried in the acknowledgment field is valid
    - Acknowledgment – segment has been successfully received
- RST, SYN, FIN – used for connection setup and teardown
- CWR, ECE – explicit congestion notification
- PSH – receiver should pass the data to the upper lay immediately
- URG – data in the segment that the sending-side upper-layer has marked as “urgent”

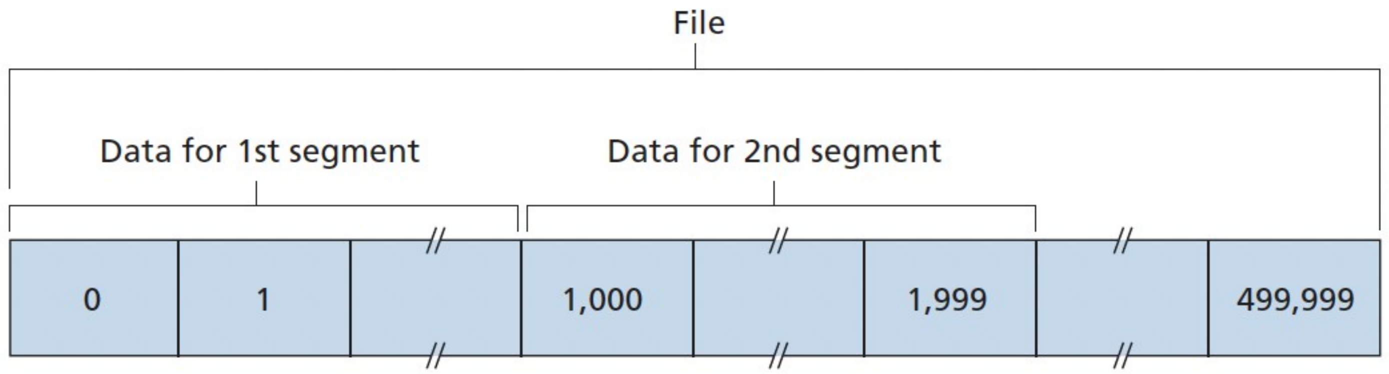
Figure 3.29 – TCP Segment Structure



Sequence Numbers and Acknowledgment Numbers

- Critical part of TCP’s reliable data transfer service
- Background
  - TCP views data as an unstructured, but ordered, stream of bytes
  - Use of sequence numbers reflects this view in that sequence numbers are over the stream of transmitted bytes and not over the series of transmitted segments
  - Sequence number for a segment – byte-stream number of the first byte in the segment
- Acknowledgment Numbers
  - Recall that TCP is full-duplex
  - Each of the the segments that arrive from Host B has a sequence number for the flowing from B to A
  - The acknowledgment number that Host A puts in its segment is the sequence number of the next byte Host A is expecting from Host B
- TCP only acknowledges bytes up to the first missing byte in the stream – TCP is said to provide cumulative acknowledgments

Figure 3.30 – Dividing File Data Into TCP Segments



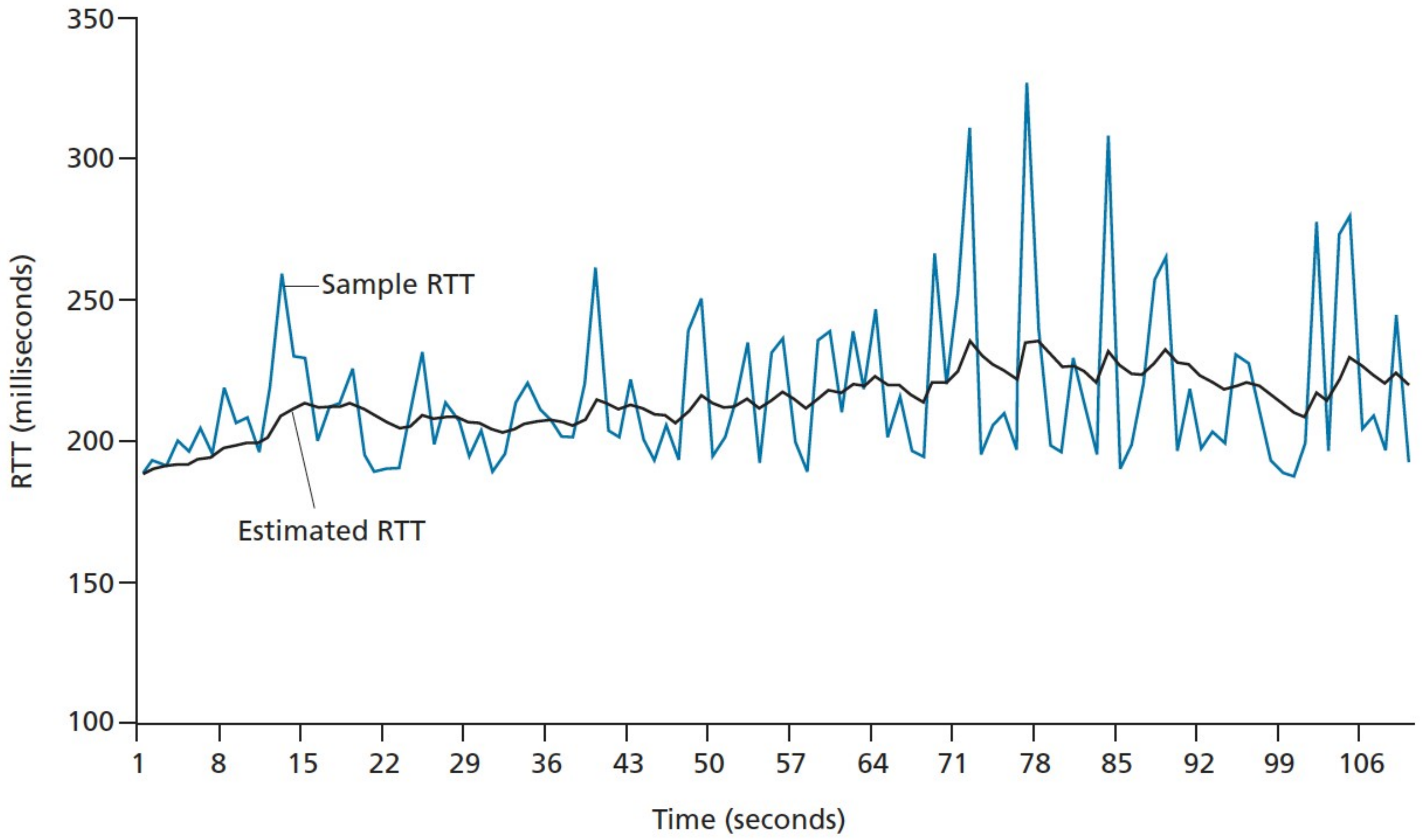
Round-Trip Time Estimation and Timeout

- TCP uses a timeout/retransmit mechanism to recover from lost segments
- Question: length of the timeout intervals?
    - Should be larger than the connection's RTT
    - But how much larger?
- Estimating Round-Trip Time
    - The sample RTT for a segment is the amount of time between when the segment is sent (passed to IP) and when an acknowledgment for the segment is received
    - Most TCP implementations take one SampleRTT measurement at a time
        - Transmitted but unacknowledged segments – leads to a new value for SampleRTT approximately once every RTT
        - TCP never computes a SampleRTT for a segment that has been retransmitted
    - SampleRTT value will fluctuate from one segment to the next
        - Congestion, varying load on the end systems

Round-Trip Time Estimation and Timeout

- Take an average of the SampleRTT values
  - TCP maintains an average, called Estimated RTT of the SampleRTT values
  - Upon obtaining a new SampleRTT, TCP updates Estimated RTT according to the following formula
    - EstimatedRTT = (1-alpha) * EstimatedRTT + alpha * SampleRTT
    - New value of EstimatedRTT is a weighted combination of the previous value of EstimatedRTT
    - Recommended value of alpha is 0.125 (1/8)
  - More weight on recent samples than on old ones
  - Figure 3.32 – SampleRTT values and EstimatedRTT for a value of alpha = 1/8 for a TCP connection between gaia.cs.umass.edu to fantasia.eurecom.fr
- Also valuable to have a measure of the variability of the RTT
  - DevRTT = (1-beta) * DevRTT + beta * |SampleRTT – EstimatedRTT|
  - If the Sample RTT values have little fluctuation, DevRTT will be small; otherwise, lots of variation means a larger DevRTT
    - Recommended value for beta: 0.25

Figure 3.32 – RTT Samples and Estimates



Round-Trip Estimation and Timeout

- Setting and Managing the Retransmission Timeout Interval
    - Given values of EstimatedRTT and DevRTT – the interval should be >= EstimatedRTT, or unnecessary retransmissions would be sent
    - Timeout interval shouldn't be too much larger than EstimatedRTT
        - When a segment is lost, otherwise, TCP would not quickly retransmit the segment
            - Large data transfer delays
    - Desirable to set timeout equal to EstimatedRTT plus some margin
    - Margin should be large when there is a lot of fluctuation in the SampleRTT values and small where there is little fluctuation
        - Value of DevRTT should come into play here
    - Timeout Interval = EstimatedRTT + 4 * DevRTT
        - Initial timeout interval value of 1 sec is recommended
        - When timeout occurs, TimeoutInterval is doubled to avoid a premature timeout occurring for a subsequent segment that will soon be acknowledged

Reliable Data Transfer

- Interesting Scenarios
- Doubling Timeout Interval
- Fast Retransmit
- Go-Back-N or Selective Repeat?

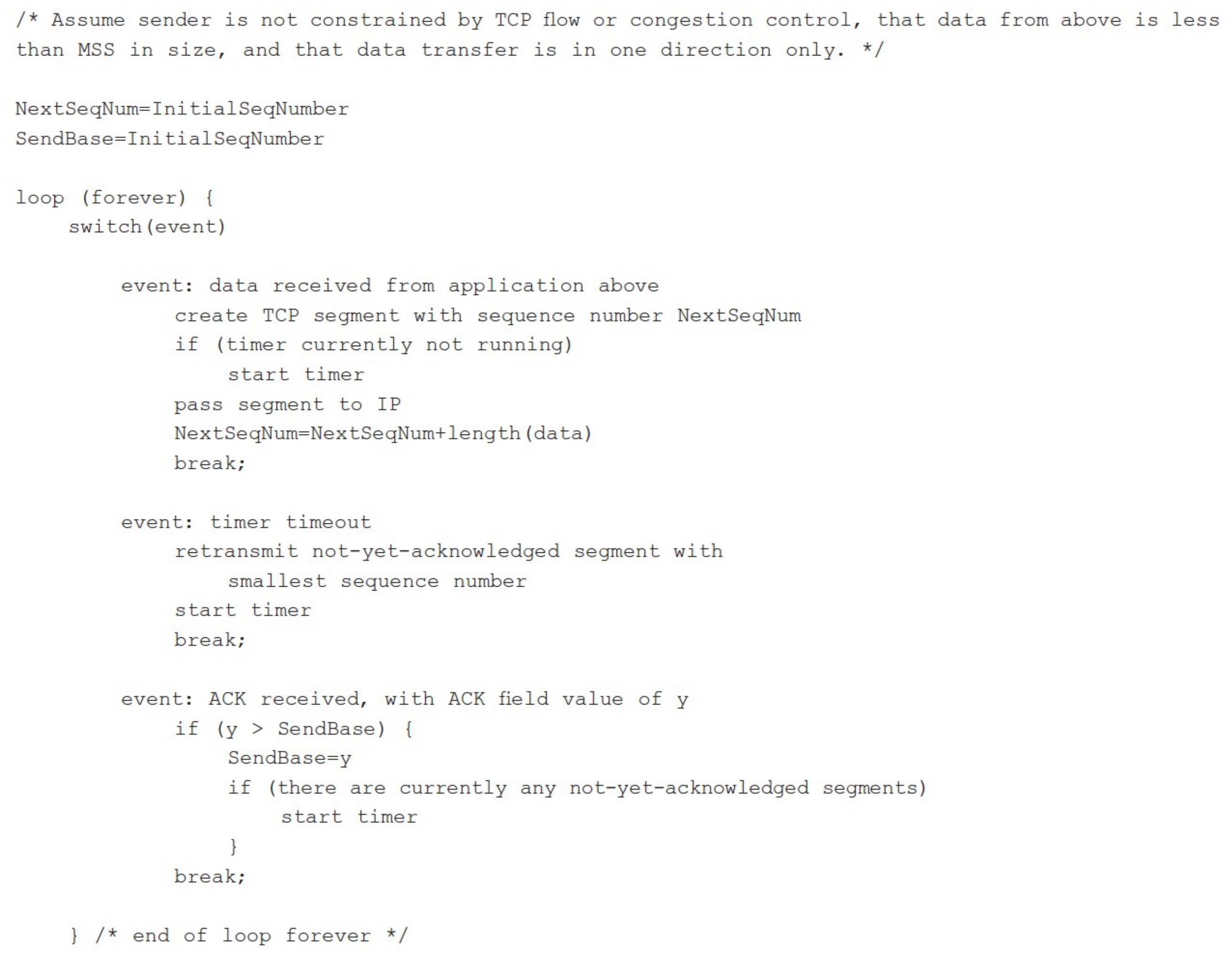
Reliable Data Transfer

- Created by TCP, on top of IP’s unreliable best-effort service
  - Ensures that data stream with a process reads out of its TCP receive buffer is uncorrupted, without gaps, without duplication, and in sequence
    - Same byte stream that was sent by the end system on the side of the connection
- How does it provide reliable data transfer?
  - “Simplified” description of a TCP sender that uses only timeouts to recover from lost segments
  - More complete description that uses duplicate acknowledgments in addition to timeouts
  - Assumption: data is being sent in one direction, from Host A to Host B; Host A is sending a large file

Reliable Data Transfer

- Figure 3.33 – simplified description of a TCP sender
    - Three major events related to data transmission and retransmission in the TCP sender
        - Data received from application above
        - Timer timeout
        - ACK receipt
    - Occurrence of first major event – TCP receives data from the application, encapsulates the data in a segment and passes the segment to IP
        - Each segment includes a sequence number that is the byte-stream number of the first data byte in the segment
        - If timer isn't running, TCP starts the timer when the segment is passed to IP
        - Expiration interval for this timer is the TimeoutInterval (shown earlier)

Figure 3.33 – Simplified TCP Sender



```
/* Assume sender is not constrained by TCP flow or congestion control, that data from above is less
than MSS in size, and that data transfer is in one direction only. */

NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber

loop (forever) {
    switch(event)

        event: data received from application above
            create TCP segment with sequence number NextSeqNum
            if (timer currently not running)
                start timer
            pass segment to IP
            NextSeqNum=NextSeqNum+length(data)
            break;

        event: timer timeout
            retransmit not-yet-acknowledged segment with
                smallest sequence number
            start timer
            break;

        event: ACK received, with ACK field value of y
            if (y > SendBase) {
                SendBase=y
                if (there are currently any not-yet-acknowledged segments)
                    start timer
                }
            break;

    } /* end of loop forever */
```

Reliable Data Transfer

- Figure 3.33
  - Second major event – timeout
    - TCP responds to the timeout event by retransmitting the segment that caused the timeout
    - TCP then restarts the timer
  - Third major event – arrival of an acknowledgment segment (ACK) from the receiver
    - TCP compares the ACK value y with its variable SendBase
      - SendBase: sequence number of the oldest unacknowledged byte
    - TCP uses cumulative acknowledgments so that y acknowledges the receipt of all bytes before byte number y
    - If y > SendBase – ACK is acknowledging one or more previously unacknowledged segments
      - Sender updates SendBase variable
      - Restarts the timer if there are any not-yet-acknowledged segments

A Few Interesting Scenarios

- Figure 3.34 – Host A sends one segment to Host B
    - Segment has sequence number 92 and contains 8 bytes of data
    - After sending, Host A waits for a segment from B, with acknowledgment number 100
    - Acknowledgment from B to A is lost even though B receives the segment
    - Timeout occurs and Host A retransmits the same segment
        - Host B receives the retransmission; TCP in Host B will discard the bytes in the retransmitted segment
- Figure 3.35 – Host A sends two segments back to back
    - First segment – sequence number 92, 8 bytes of data
    - Second segment has sequence number 100 and 20 bytes of data
    - Both segments arrive intact at B and B sends two separate acknowledgments for each of these segments
    - First acknowledgment – has number 100; second – 120
    - Neither of the acknowledgments arrives at A before the timeout
    - Timeout occurs, A resends the first segment with sequence number 92 and restarts the timer
        - As long as the ACK for the second segment arrives before the new timeout, the second segment will not be retransmitted

A Few Interesting Scenarios

- Last scenario – same as the second, but before the timeout event, A receives an acknowledgment with number 120
  - A knows that Host B has received everything up through byte 119, so it does not need to resend either of the two segments
  - Figure 3.36

Figure 3.34 – Retransmission Due to a Lost Acknowledgment

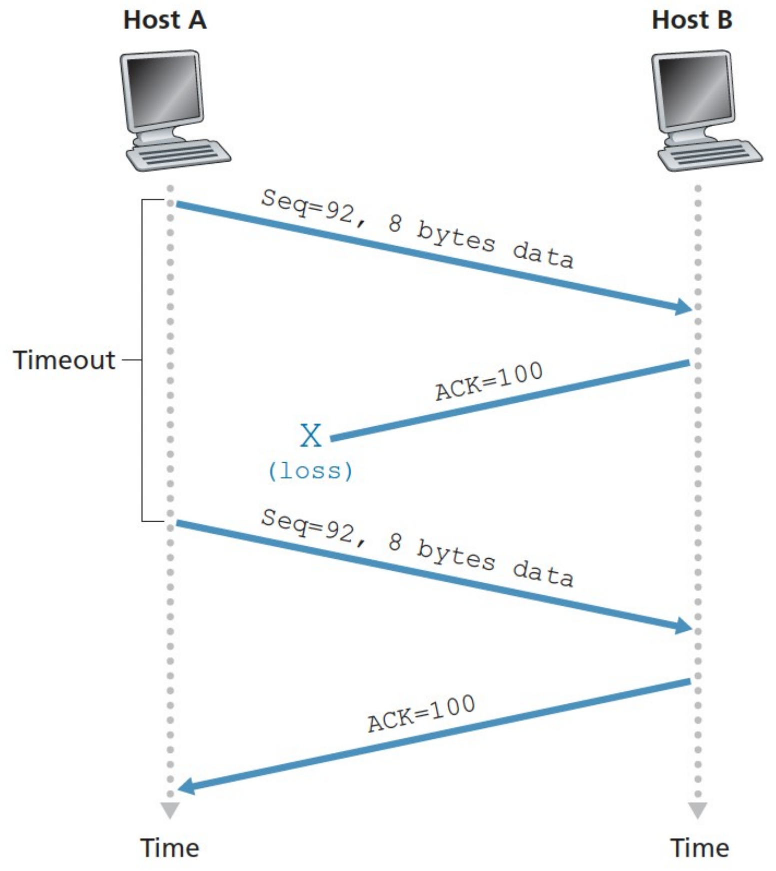


Figure 3.35 – Segment 100 Not Retransmitted

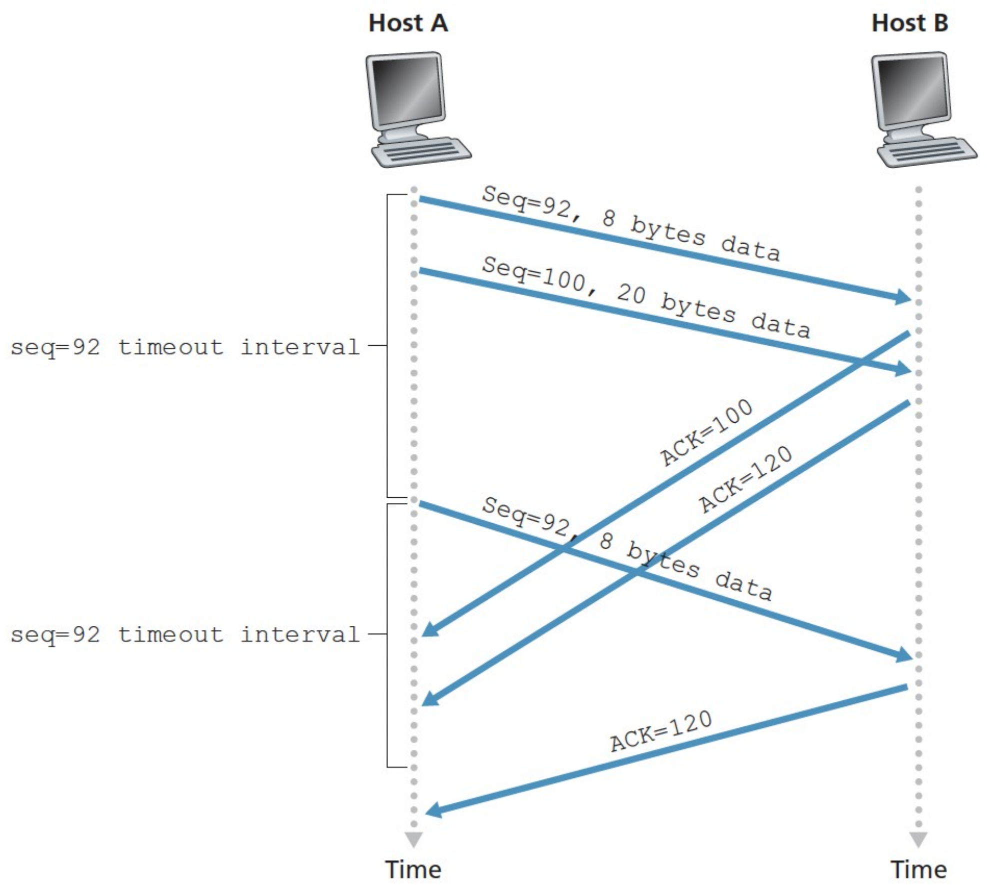
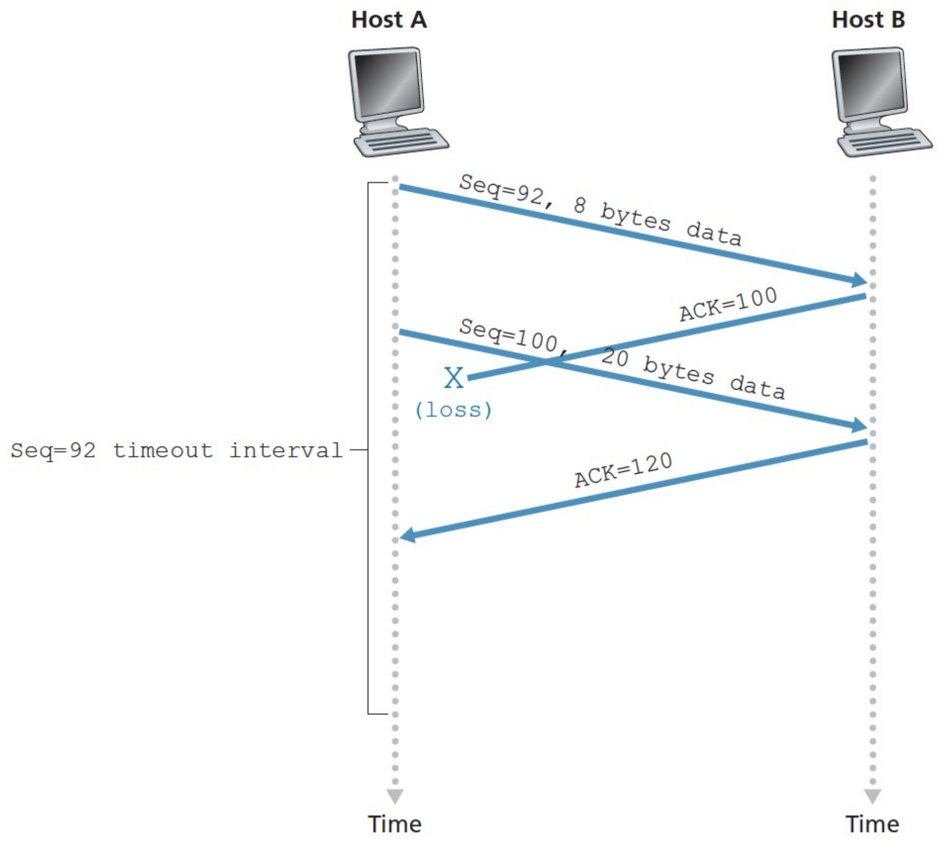


Figure 3.36 – A Cumulative Acknowledgment Avoids Retransmission of the First Segment



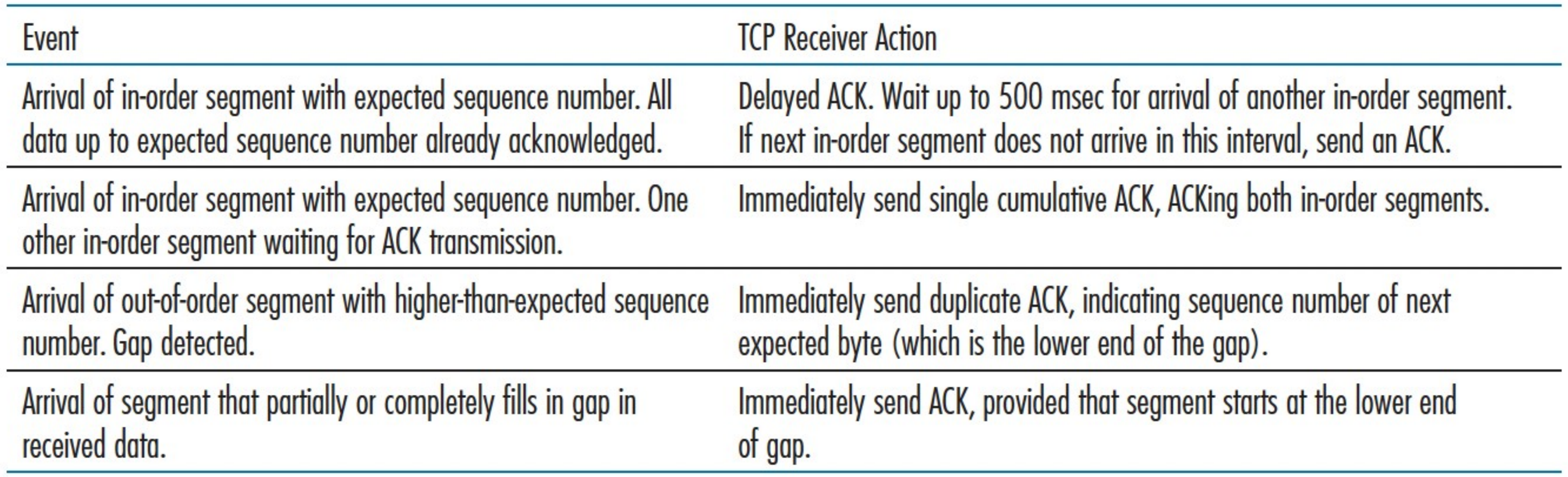
Doubling the Timeout Interval

- Modifications that most TCP implementations use
  - Length of the timeout interval after a timer expiration
    - Whenever the timeout event occurs, TCP retransmits the not-yet-acknowledged segment with the smallest sequence number
    - Each time TCP retransmits, it sets the next timeout interval to twice the previous value rather than deriving from our previous calculations
- Example
  - Oldest, not yet acknowledged segment TimeoutInterval: .75 sec when it expires
  - TCP retransmits segment and sets new expiration time to 1.5 sec
  - If timer expires again, TCP will again retransmit the segment, setting expiration time to 3.0 sec
- Provides a form of congestion control
  - Timer expiration is (most likely) caused by network congestion
  - Instead of bombarding at the same interval each time (like other packets that might be sent elsewhere), TCP gives longer intervals

Fast Retransmit

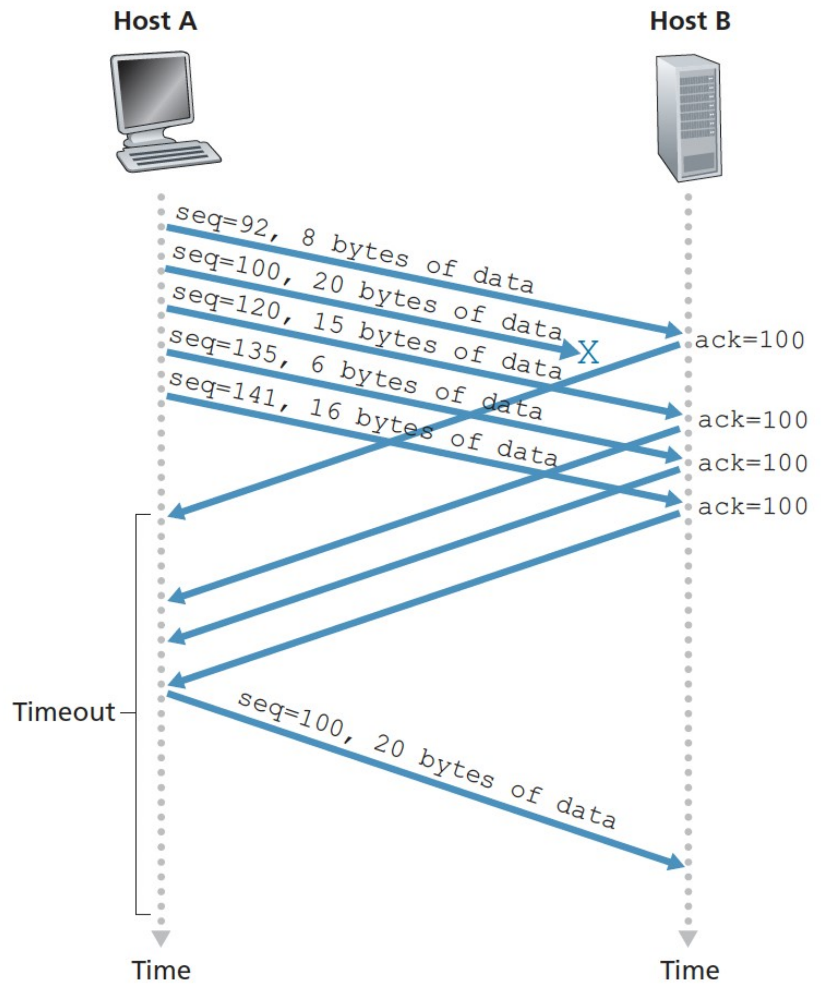
- A problem with timeout-triggered retransmissions – timeout period can be relatively long
    - When a segment is lost, long timeout period forces the sender to delay resending the lost packet – increases end-to-end delay
    - Sender can often detect packet loss well before timeout event occurs by noting duplicate ACKs
        - ACK that acknowledges a segment for which the sender has already received an earlier ACK
- Table 3.2 – Receiver's ACK generation policy
    - When a TCP receiver gets a segment with a sequence number that is larger than the next, expected, in-order sequence number – gap in data stream
        - Missing segment
        - Since TCP does not utilize negative ACKs, receiver cannot send such a message back to the sender; reacknowledges the last in-order byte of data it has received
- Because a sender often sends a large number of segments back-to-back, if one segment is lost, there will likely be many back-to-back duplicate ACKs
    - If the TCP sender receives three duplicate ACKS for the same data, it takes this as an indication that the segment following the segment that has been ACKed three times has been lost
    - In this case, the TCP sender performs fast retransmit – before that segment's timer expires

Table 3.2 – TCP ACK Generation Recommendation (RFC 5681)



Event	TCP Receiver Action
Arrival of in-order segment with expected sequence number. All data up to expected sequence number already acknowledged.	Delayed ACK. Wait up to 500 msec for arrival of another in-order segment. If next in-order segment does not arrive in this interval, send an ACK.
Arrival of in-order segment with expected sequence number. One other in-order segment waiting for ACK transmission.	Immediately send single cumulative ACK, ACKing both in-order segments.
Arrival of out-of-order segment with higher-than-expected sequence number. Gap detected.	Immediately send duplicate ACK, indicating sequence number of next expected byte (which is the lower end of the gap).
Arrival of segment that partially or completely fills in gap in received data.	Immediately send ACK, provided that segment starts at the lower end of gap.

Figure 3.37 – Fast Retransmit – Retransmitting the Missing Segment Before the Segment’s Timer Expires



GBN or SR?

- Is TCP a GBN or SR?
  - Based on prior descriptions, it looks like a GBN-style protocol
- But – there are differences between TCP and GBN
  - Many TCP implementations will buffer correctly received but out-of-order segments
  - What happens when the sender sends a sequence of segments and all of the segments arrive in order without error at the receiver
  - Also, acknowledgment for packet $n < N$ gets lost, but all remaining acknowledgments arrive at the sender before timeout
  - GBN would retransmit not only packet n, but all other packets that came after n
  - TCP – retransmits at most one segment, n.
    - Would not retransmit if acknowledgment for segments for n+1, n+2, etc., are received

GBN or SR?

- TCP – proposed modification – Selective Acknowledgment
  - Allows TCP receiver to acknowledge out-of-order segments selectively rather than cumulatively acknowledging the last correctly received, in-order segment
  - When combined with selective retransmission – TCP looks like the generic SR protocol
- As a result: TCP is a “hybrid” of GBN and SR