Transport Layer

Instructor: C. Pu (Ph.D., Assistant Professor)

Lecture 11

puc@marshall.edu

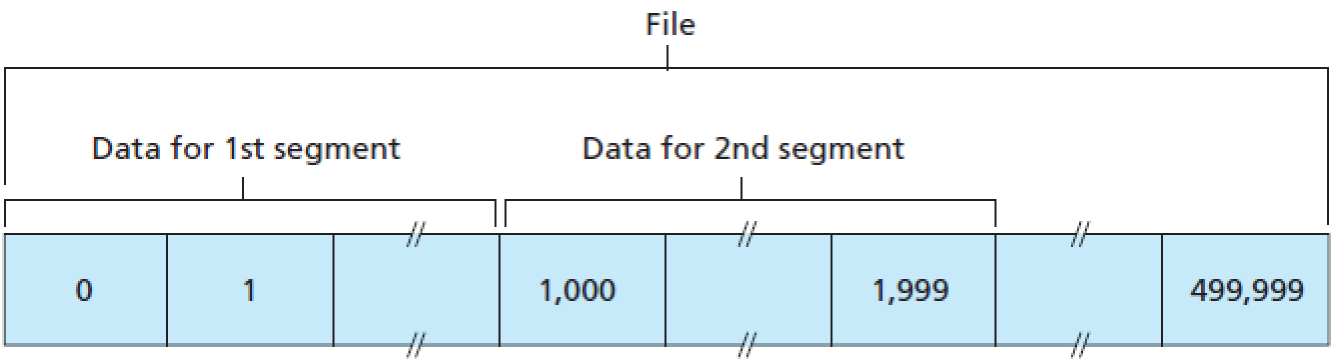
TCP Seq. #'s and ACKs (e.g., TELNET)

- two of the most important fields in the TCP segment header:
  - *sequence number field*
  - *acknowledgment number field*

- TCP views data as an *unstructured*, but ordered, stream of bytes
  - *sequence numbers are over the stream of transmitted bytes*
    - not over the series of transmitted segments
  - thus, the sequence number for a segment is the byte-stream number of the ***first byte*** in the segment.

TCP Seq. #'s and ACKs (e.g., TELNET)

- Seq. #:
  - byte stream "number" of *first byte* in segment's data
    - not over the series of transmitted segments
  - e.g. a file consisting of 500,000 bytes, and MSS (1,000 bytes)
    - 500 segments out of the data stream
    - 1st segment's sequence # : **0**
    - 2nd segment's sequence # : **1,000**
    - 3rd segment's sequence # : **2,000**, and so on



TCP Seq. #'s and ACKs (e.g., TELNET)

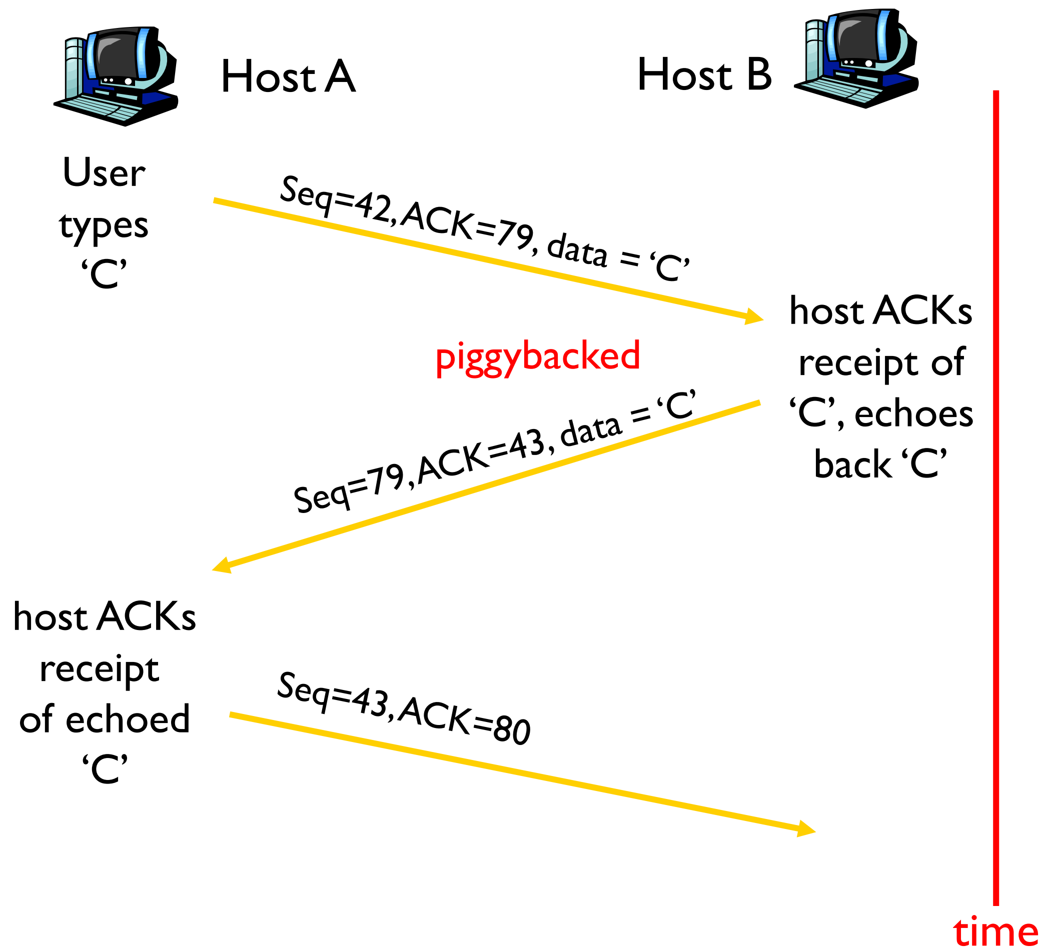
- Ack. #:
  - sequence # of the next byte of data that the host is waiting for
  - ***cumulative ACK***:
    - only ack bytes up to the *first missing byte* in the stream
  - example:
    - host A has received all bytes numbered 0 through 535 from B and suppose that it is about to send a segment to host B
    - host A is waiting for byte 536 and all the subsequent bytes in host B's data stream
    - so host A puts 536 in the acknowledgment number field of the segment it sends to B
      - ***536*** is ***the next byte of data*** the host A is waiting for

TCP Seq. #'s and ACKs (e.g., TELNET)

- Ack. #:
  - sequence # of the next byte of data that the host is waiting for
  - ***cumulative ACK***:
    - only ack bytes up to the first missing byte in the stream
  - another example:
    - host A has received one segment from host B containing bytes 0 through 535 and another segment containing bytes 900 through 1,000
    - for some reason host A has not yet received bytes 536 through 899
    - host A is still waiting for byte 536 (and beyond) in order to re-create B's data stream
    - A's next segment to B will contain 536 in the acknowledgment number field
    - because TCP only acknowledges bytes up to the *first missing byte* in the stream, TCP is said to provide ***cumulative acknowledgments***

TCP Seq. #'s and ACKs (e.g., TELNET)

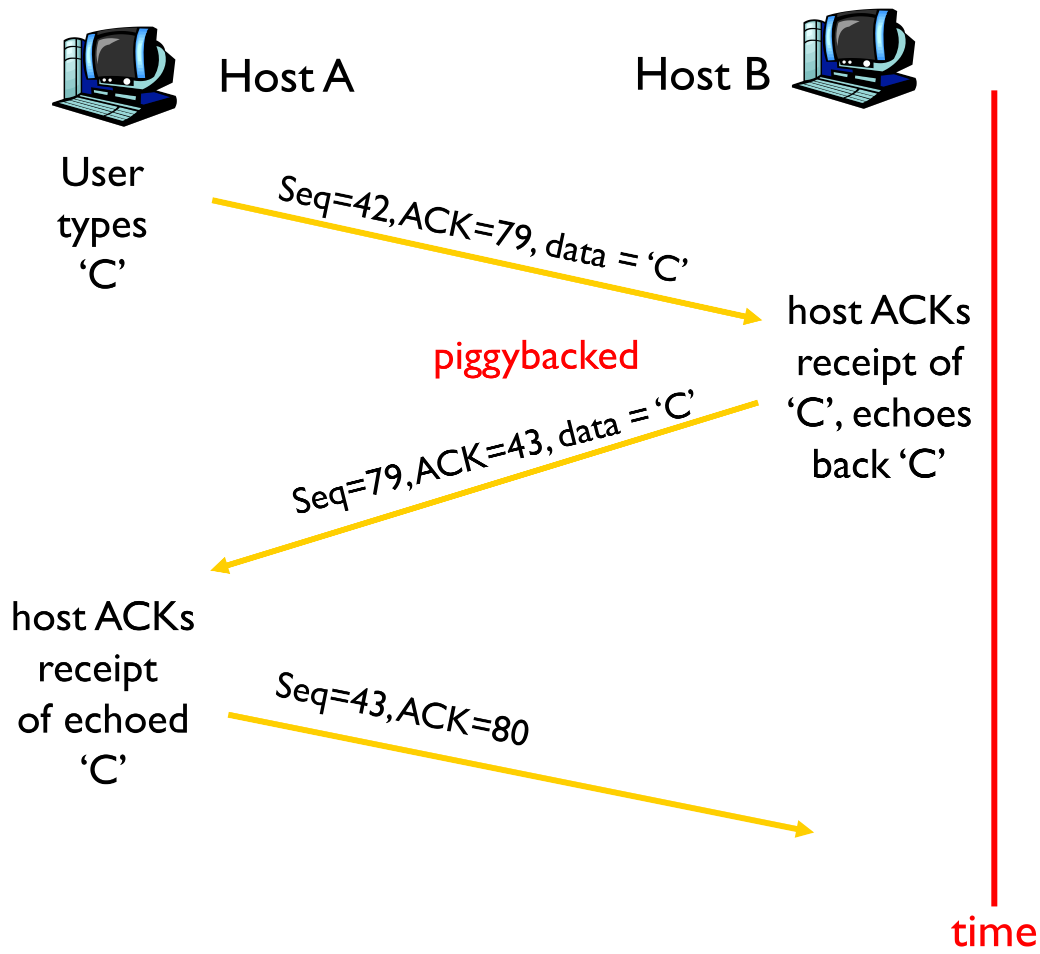
- Telnet (RFC 854)
  - application-layer protocol used for remote login
  - runs over TCP
  - work between any pair of hosts
  - Telnet is an interactive application
    - nicely illustrates TCP sequence and acknowledgment numbers



simple telnet scenario

TCP Seq. #'s and ACKs (e.g., TELNET)

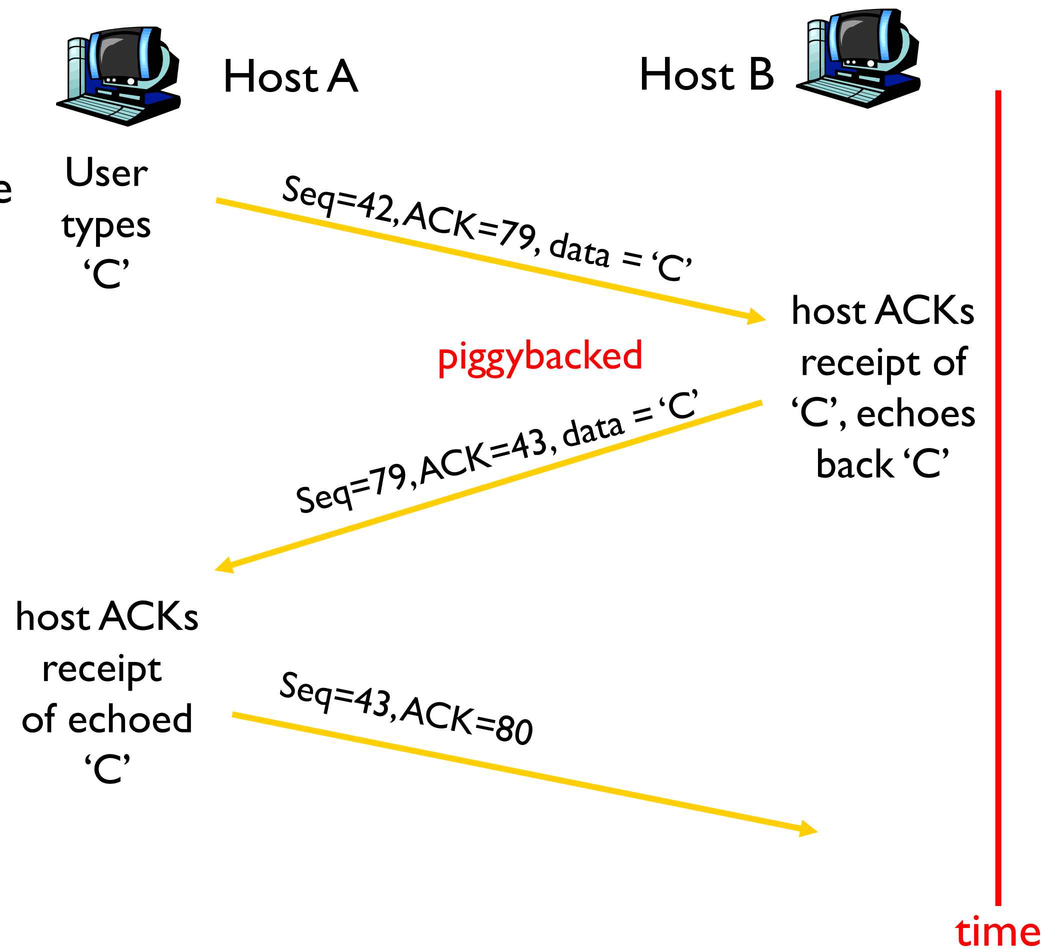
- Example:
  - host A initiates a Telnet session with host B
  - host A: client
  - host B: server
  - each character typed by the user will be sent to the remote host; the remote host will send back a copy of each character, which will be displayed on the Telnet user's screen
  - assuming that starting sequence numbers are 42 and 79 for the client and server.



simple telnet scenario

TCP Seq. #'s and ACKs (e.g., TELNET)

- Seq. #'s:
  - byte stream "number" of first byte in segment's data
- ACKs:
  - seq # of next byte expected from other side
  - cumulative ACK



simple telnet scenario

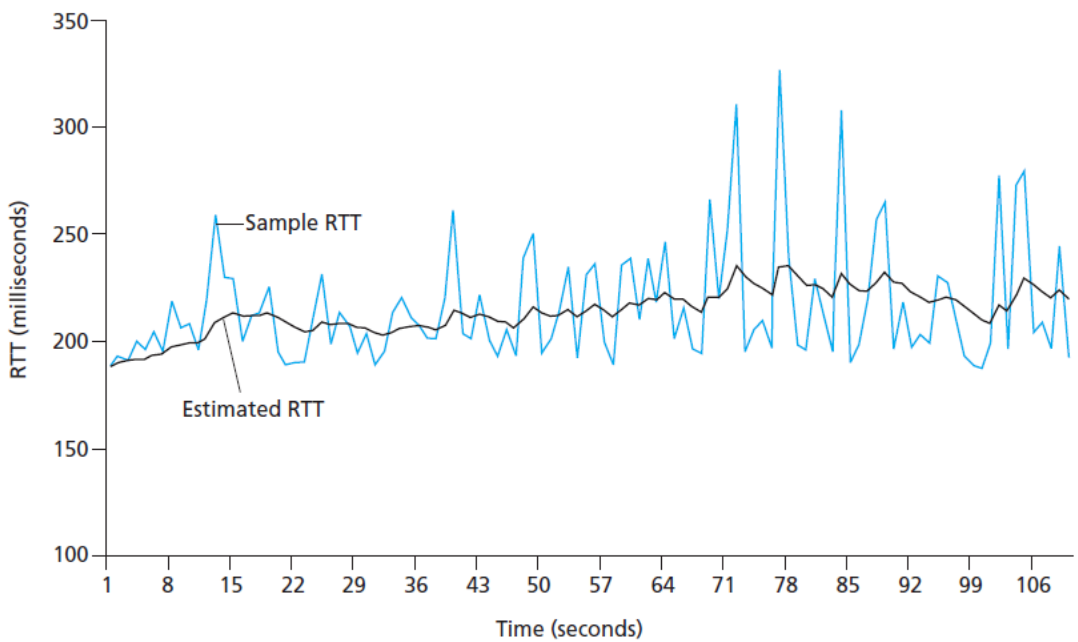
TCP Round Trip Time (RTT) and Timeout

- TCP uses *timeout/retransmit* mechanism to recover from lost segments
- Q: how to set TCP timeout value?
  - longer than RTT
    - the time from when a segment is sent until it is acked
    - but RTT varies
  - too short: premature timeout
    - unnecessary retransmissions
  - too long: slow reaction to segment loss
- Q: how to estimate RTT?
  - **SampleRTT**: measured time from segment transmission until ACK receipt
    - ignore retransmissions
  - **SampleRTT** will vary, want estimated RTT "smoother"
    - average several recent measurements, not just current **SampleRTT**

TCP Round Trip Time (RTT) and Timeout (cont.)

- EstimatedRTT = (1- α) * EstimatedRTT + α * SampleRTT
  - the new value of EstimatedRTT is a weighted combination of the previous value of EstimatedRTT and the new value for SampleRTT
  - typical value: α = 0.125

- Example RTT estimation:



TCP Round Trip Time (RTT) and Timeout (cont.)

- In addition to having an estimate of the RTT, it is also valuable to have a measure of *the variability of the RTT*

- RTT variation: DevRTT, as an estimate of how much SampleRTT typically deviates from EstimatedRTT

$$DevRTT = (1 - \beta) * DevRTT + \beta * |SampleRTT - EstimatedRTT|$$

  - β is 0.25

TCP Round Trip Time (RTT) and Timeout (cont.)

- given values of EstimatedRTT and DevRTT, what value should be used for TCP's timeout interval?
  - the interval should be greater than or equal to EstimatedRTT, or unnecessary retransmissions would be sent
  - but the timeout interval should not be too much larger than EstimatedRTT
- desirable to set the timeout equal to the EstimatedRTT plus some margin
  - the margin should be large when there is a lot of fluctuation in the SampleRTT values
  - it should be small when there is little fluctuation
  - the value of DevRTT should thus come into play here
- then set timeout interval:
  - TimeoutInterval = EstimatedRTT + 4 * DevRTT

TCP Reliable Data Transfer

- TCP creates ***rdt*** service on top of IP's *unreliable service*
  - pipelined segments
  - cumulative acks
  - TCP uses single retransmission timer

- retransmissions are triggered by:
  - timeout events
  - duplicate acks

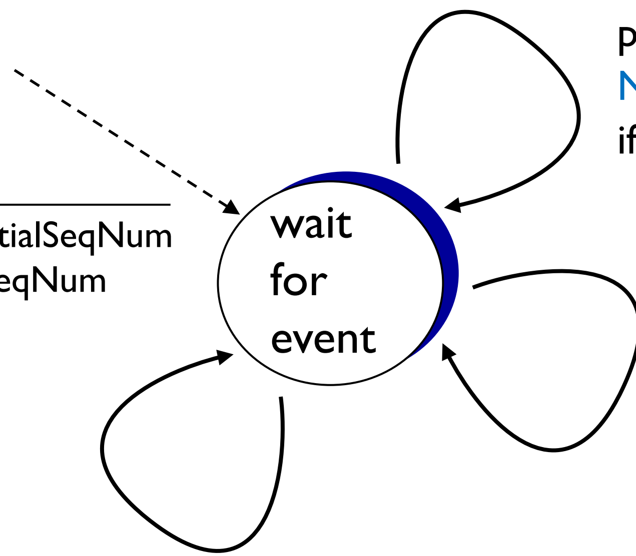
TCP Reliable Data Transfer

- suppose that data is being sent in only one direction, from Host A to Host B, and that Host A is sending a large file
  - first present a highly simplified description of a TCP sender that uses only timeouts to recover from lost segments
  - second present a more complete description that uses duplicate acknowledgments in addition to timeouts

TCP Sender Events (cont.)

L

NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum



data received from application above

create segment, seq. #: NextSeqNum
pass segment to IP (i.e., “send”)
NextSeqNum = NextSeqNum + length(data)
if (timer currently not running)
    start timer

timeout

retransmit not-yet-acked segment
with **smallest** seq. #
start timer

ACK received, with ACK field value y

```
if (y > SendBase) {     /* SendBase is the oldest unacked byte */
    SendBase = y
    /* SendBase–1: last cumulatively ACKed byte */
    if (there are currently not-yet-acked segments)
        start timer
    else stop timer
}
```

TCP Sender Events

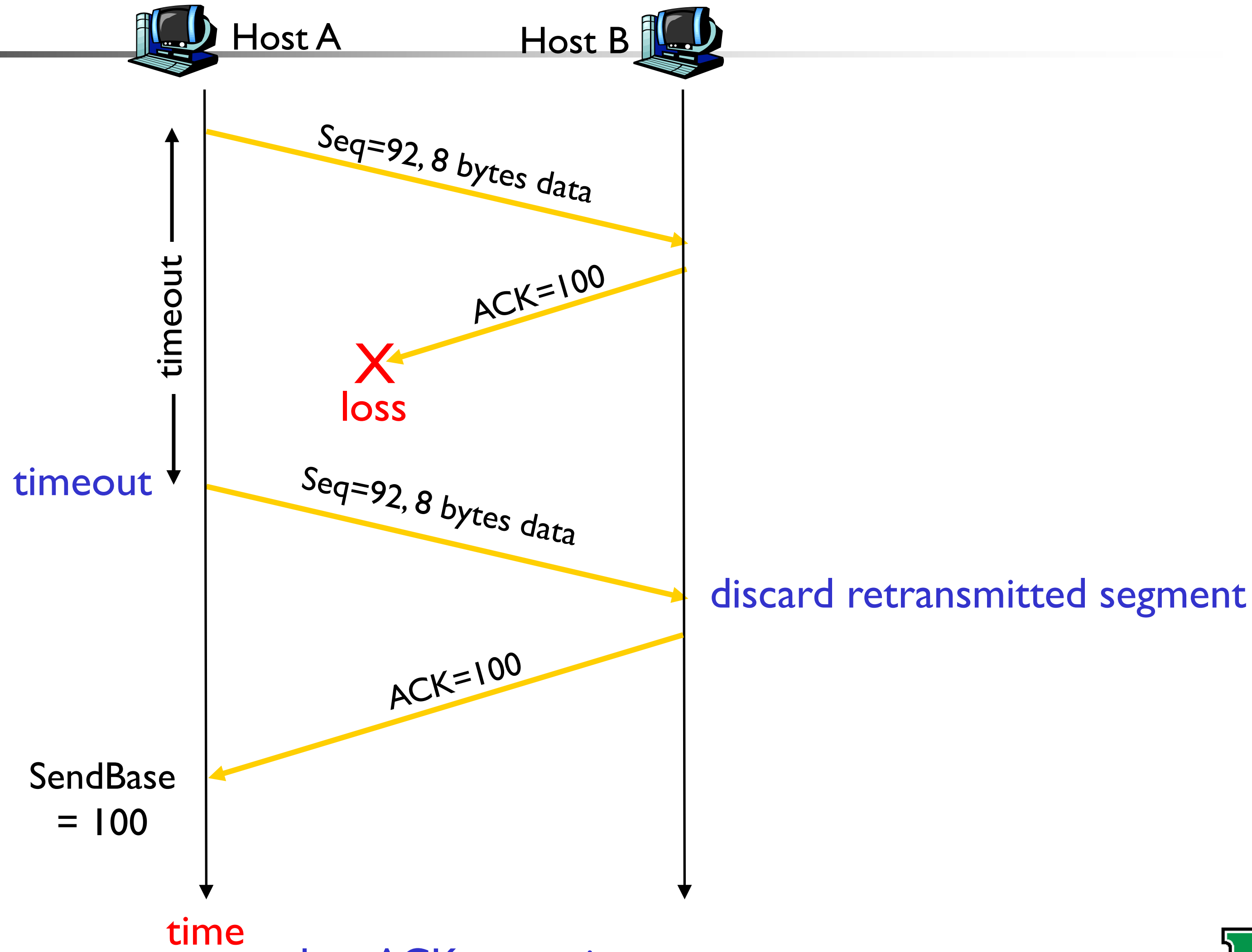
- data rcvd from app.:
  - create segment with seq #
  - seq # is byte-stream number of **first data byte** in segment
  - start timer if not already running (think of timer as for oldest unacked segment)
  - expiration interval: `TimeOutInterval`
- timeout:
  - retransmit segment that caused timeout
  - restart timer
- Ack rcvd:
  - if acknowledges previously unacked segments
    - update what is known to be acked
    - start timer if there are still unacked segments

TCP Sender Events (cont.)

- comment:
  - SendBase - 1: last cumulatively ack'ed byte

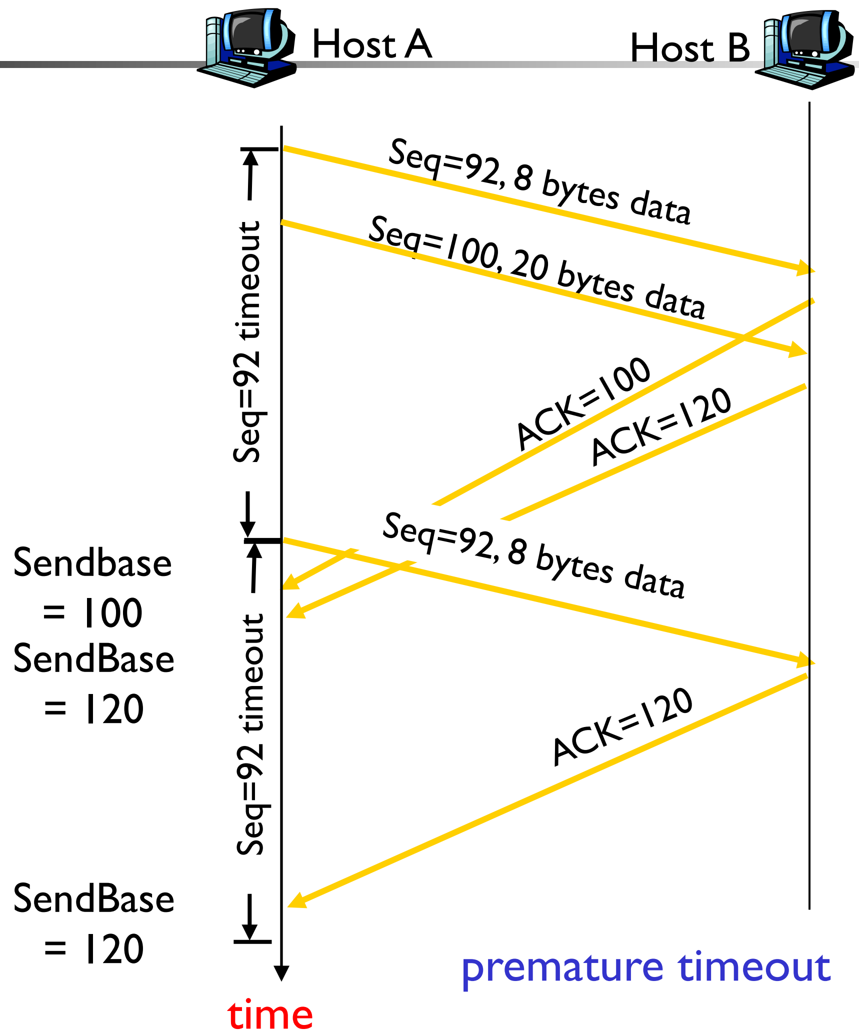
```
NextSeqNum = InitialSeqNum;
SendBase = InitialSeqNum;

loop (forever) {
  switch(event) {

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

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

   event: ACK received, with ACK field value of y
      if (y > SendBase) {
          SendBase = y;
          /* SendBase – 1: last cumulative ACKed byte */
         if (there are currently not-yet-acknowledged segments)
              start timer;
         else
              stop timer;
         }
   } /* end of switch */
}  /* end of loop forever */
```

TCP: Retransmission Scenarios

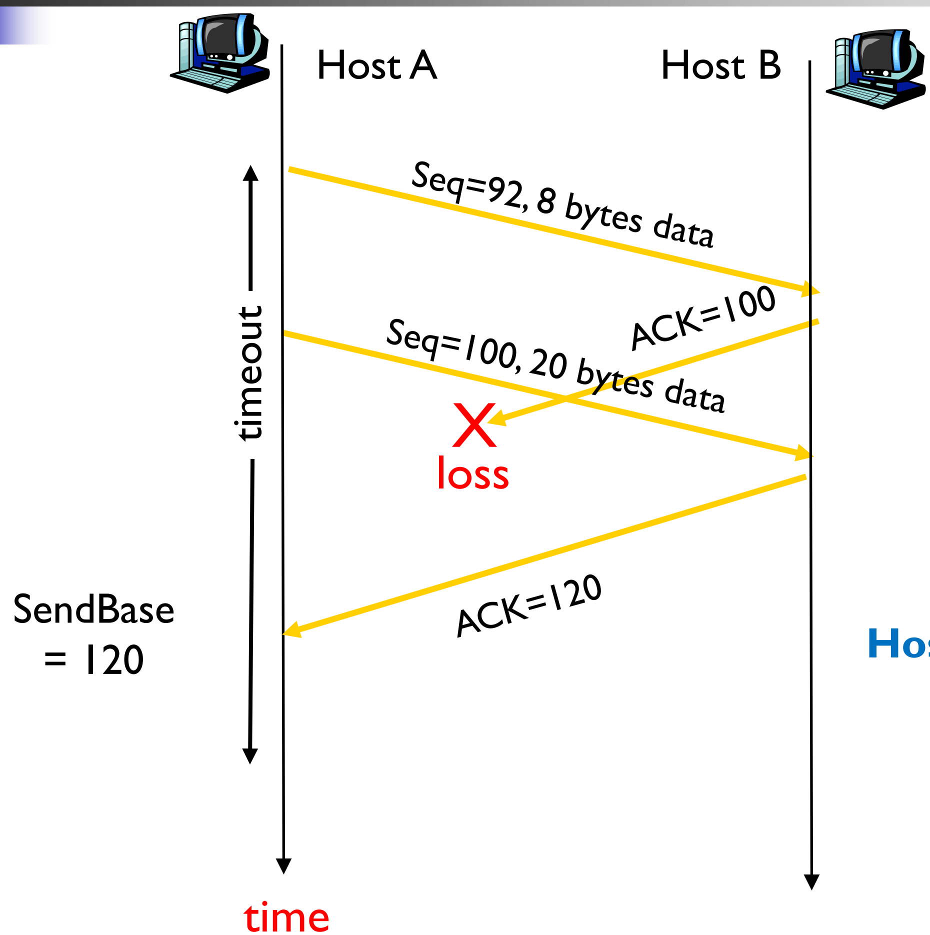


lost ACK scenario

TCP: Retransmission Scenarios



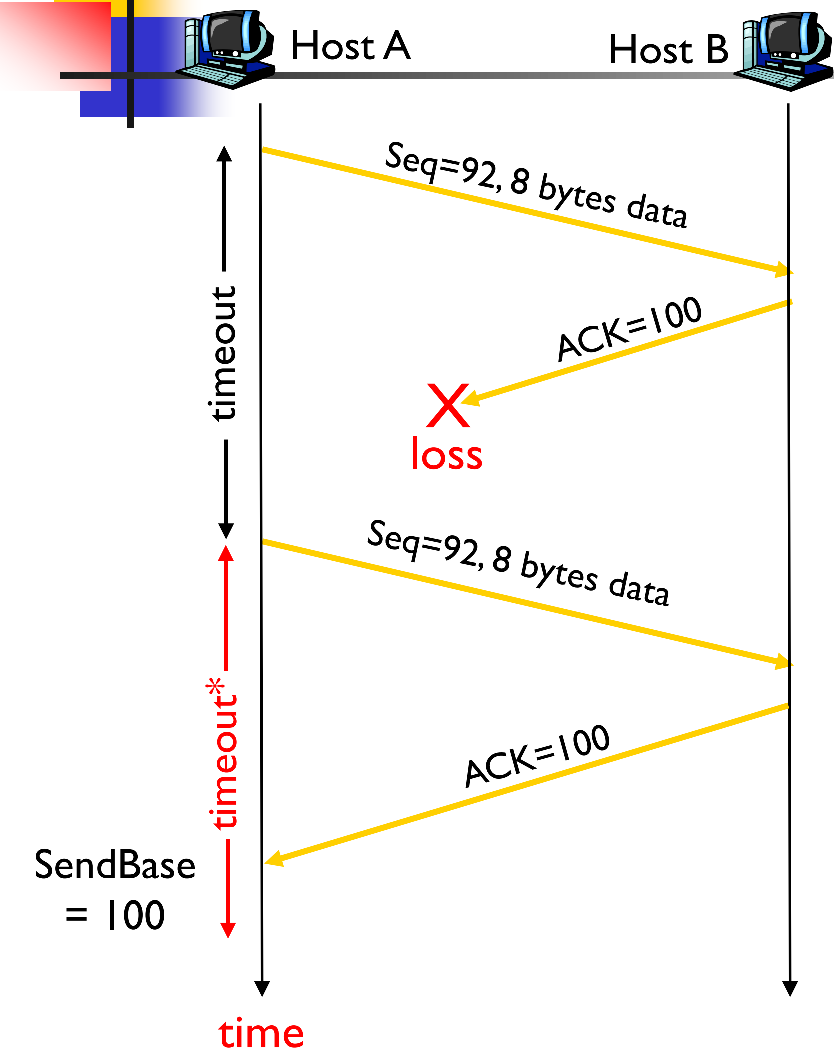
TCP: Retransmission Scenarios (cont.)



Host A knows that Host B has received everything up through byte 119!!

Cumulative ACK scenario

TCP: Retransmission Scenarios



lost ACK scenario

- doubling the timeout interval
- each time TCP retransmission
  - set the next timeout interval to twice the previous value
    - not derive the value from EstimatedRTT and DevRTT
  - e.g. 0.75, 1.5, 3.0, 6.0, etc
  - related to congestion control
- whenever the timeout event occurs,
  - retransmit the not-yet-ack segment with the smallest sequence #
- whenever the timer is started (e.g. data packet from application layer or ack received)
  - timeout value is derived from the most recent value (EstimatedRTT and DevRTT)