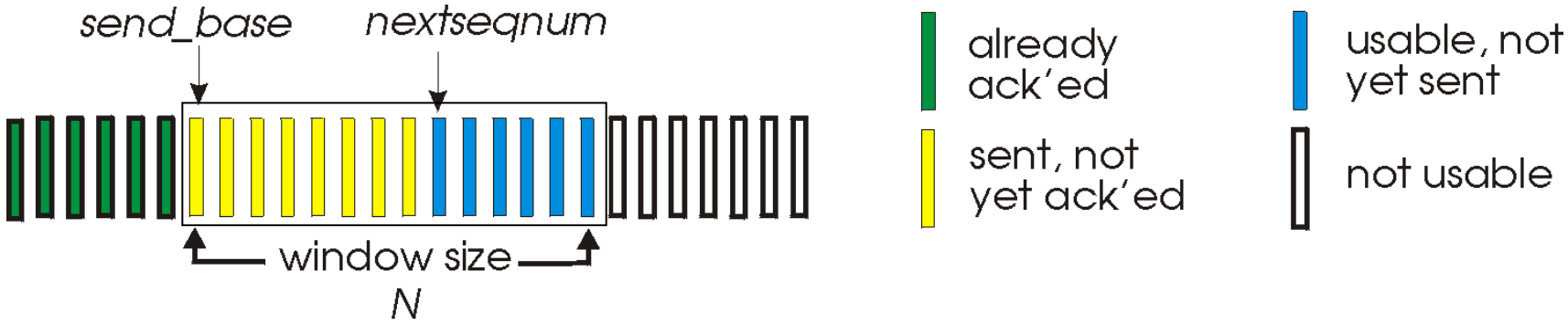
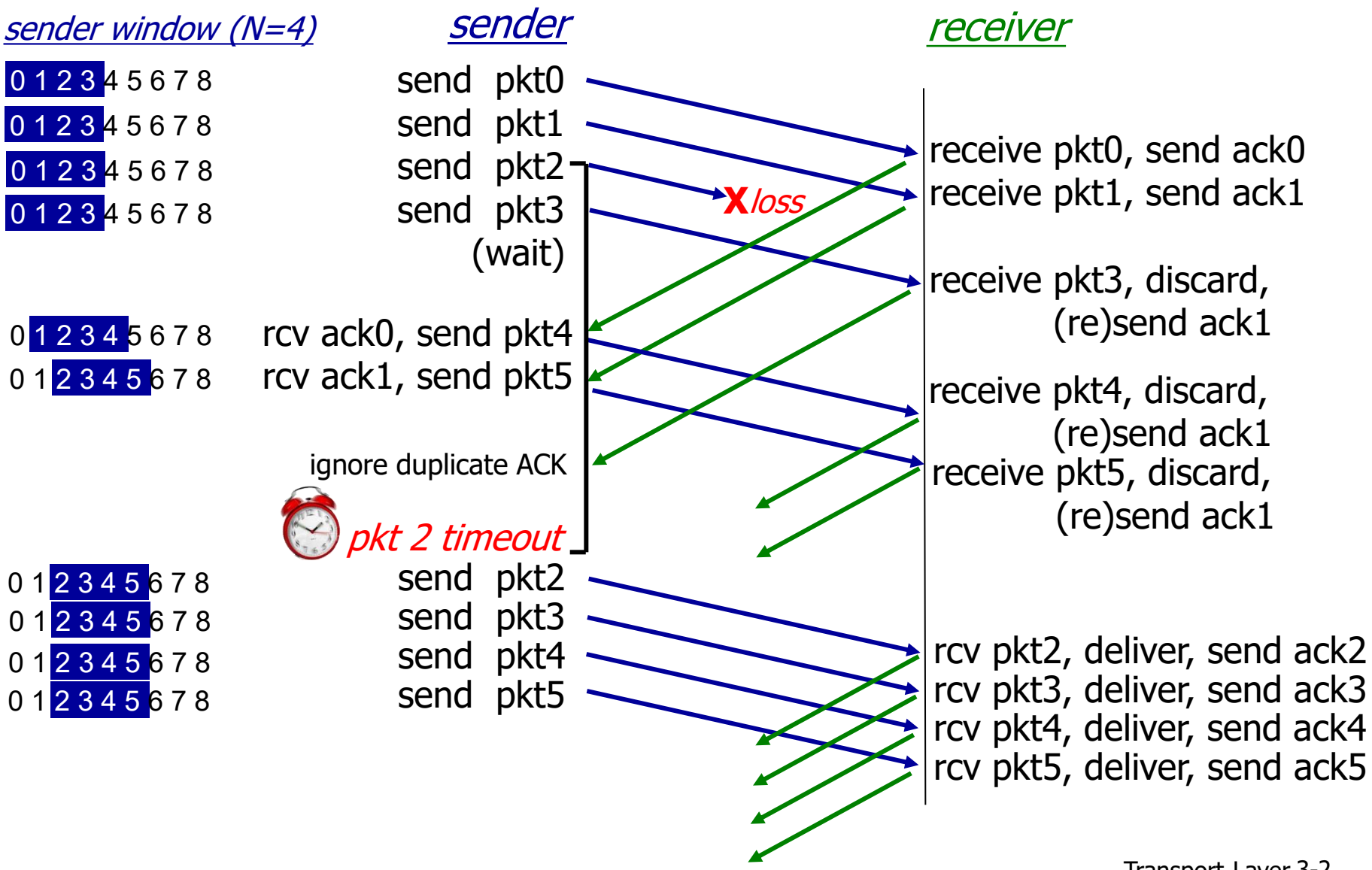
Go-Back-N: sender

1. Window: up to N consecutive unack'ed packets allowed



2. cumulative ACK(n): all packets with a sequence number up to and including n have been correctly received at the receiver
   - may receive duplicate ACKs (see receiver)
3. timer for oldest in-flight pkt (i.e., send_base)
   - *timeout*: retransmit packets from send_base to nextseqnum-1 (i.e., yellow region)

GBN in action



Selective repeat

- receiver *individually* acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt
- sender window
  - *N* consecutive seq #' s
  - limits seq #s of sent, unACKed pkts

Selective repeat

sender

data from above:

- if next available seq # in window, send pkt

timeout(n):

- resend pkt n, restart timer

ACK(n) in [sendbase,sendbase+N]:

- mark pkt n as received
- if n smallest unACKed pkt, advance window base to next unACKed seq #

receiver

pkt n in [rcvbase, rcvbase+N-1]

- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

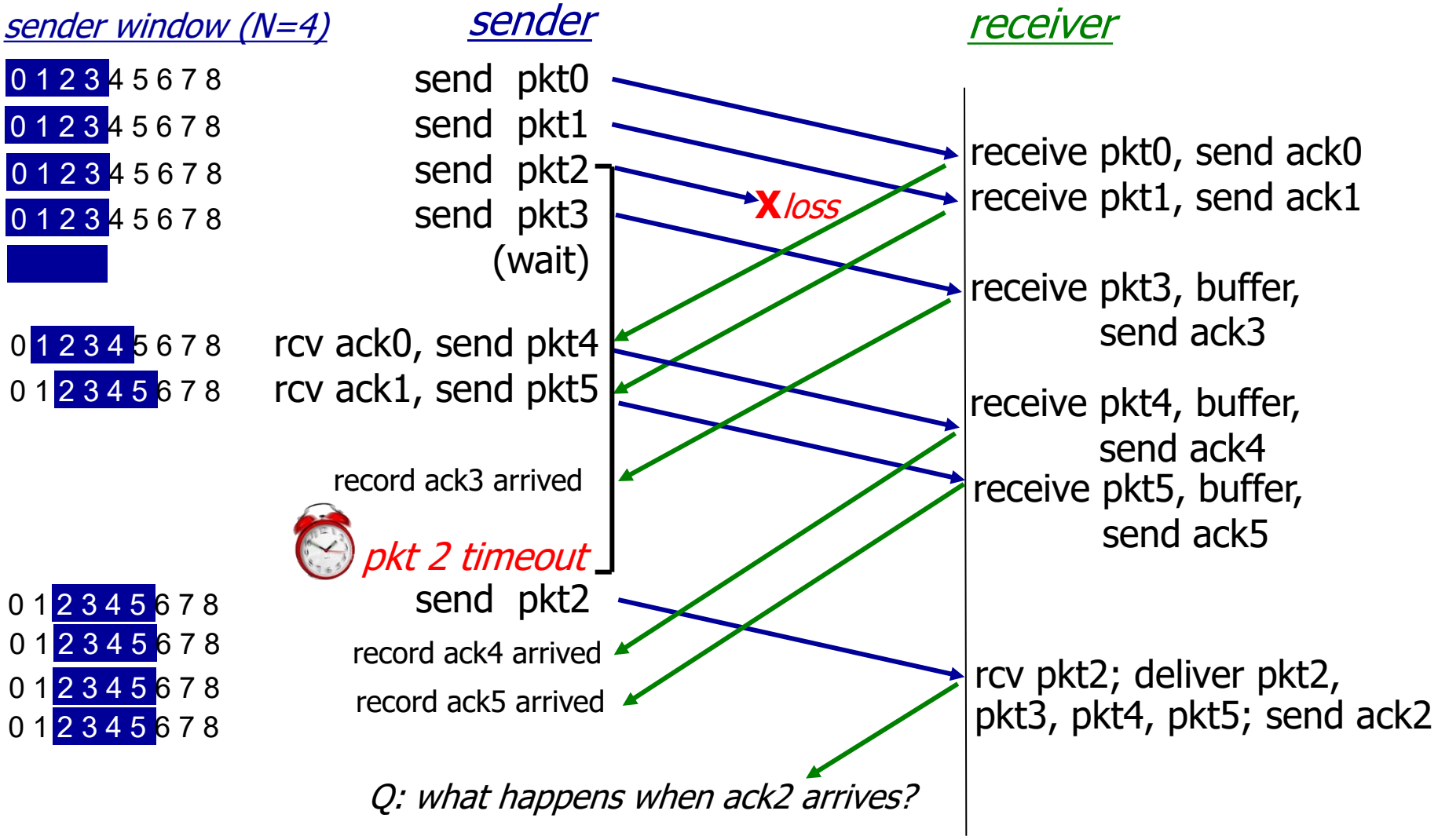
pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

otherwise:

- ignore

Selective repeat in action



Problem 1

a) For GBN, the ACK is cumulative. Thus, the ACKs that the sender will receive are

ACK0, ACK1, ACK2, ACK3, ACK4, ACK5, ACK5.

Note: For the lost pkt6, the receiver will not send any ACK back. The last ACK5 is the response of receiving pkt7.

b) For SR, only correctly received packets will be individually acknowledged, while all other cases will be ignored. Thus, the ACKs that the sender will receive are

ACK0, ACK1, ACK2, ACK3, ACK4, ACK5, ACK7.

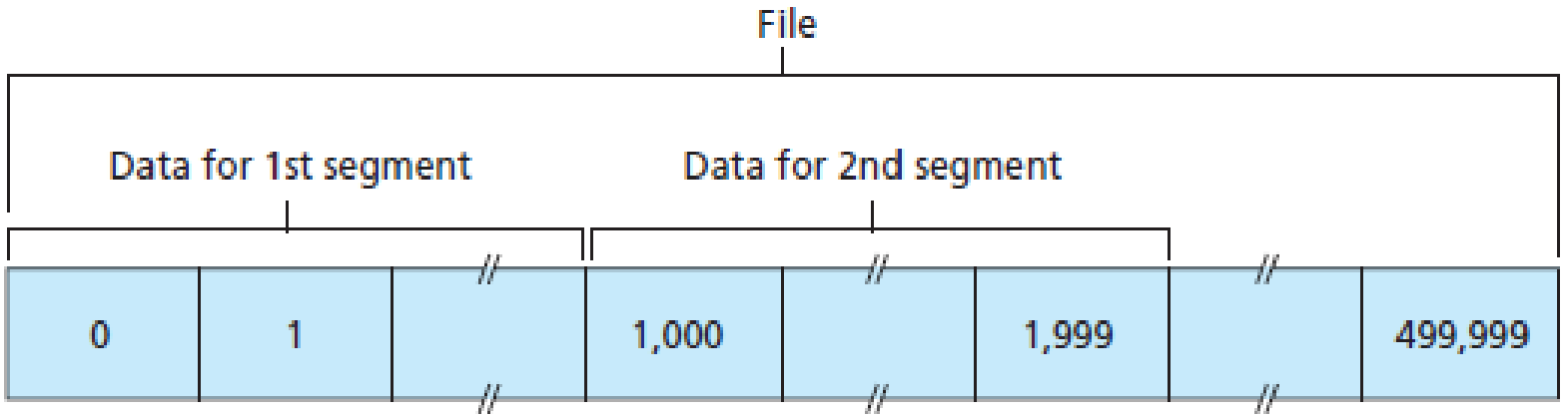
Note: since pkt7 is out-of-order, the receiver will buffer it until pkt6 is eventually correctly received (from retransmission by sender).

TCP payload

- A TCP header (typically 20-60 bytes long), which includes metadata like source/destination ports, sequence numbers (to track the order of bytes in the data stream), acknowledgment numbers (for confirming receipt), flags (e.g., SYN, ACK, FIN), and checksums for error detection.
- A payload (the actual data being transferred), which is limited in size by the Maximum Segment Size (MSS). The MSS is the maximum amount of data that can fit into one segment (excluding the header), often negotiated during connection setup to avoid fragmentation. For example, a common MSS is 1,460 bytes on Ethernet networks.

TCP seq. numbers

- Maximum segment size (MSS): A TCP payload (the actual data being transferred), which is limited in size by the Maximum Segment Size (MSS).
- Sequence number for a segment: Byte-stream number of the first byte in the segment
- E.g., file size = 500,000 bytes and MSS = 1,000 bytes



Dividing file data into TCP segments

TCP ACKs

- ACK number that receiver puts in its segment

= seq # of next byte expected from other side

  - E.g., Host A has received all bytes 0 through 535 from B

  -> ACK number = 536

  - E.g., Host A has received one segment from Host B containing bytes 0 through 535 and another segment containing bytes 900 through 1,000

  -> ACK number = 536

  (acknowledges bytes up to the first missing byte of 536)

- Cumulative ACK (GBN-style by default): TCP acknowledges bytes *up to* the first missing byte in the stream
- Q: how receiver handles out-of-order segments
  - A: TCP spec doesn't say, - up to implementor

Problem 2

- TCP treats data as a continuous byte stream, where each segment's sequence number (seq) indicates the position of its first byte in that stream.
- The acknowledgment number (ack) in a response segment tells the sender "I've received everything up to (but not including) this byte number."
- Host B acknowledges every incoming segment, but the ack number depends on whether the data is in-order.

Problem 2

- Given Setup
  - Host B has received and acknowledged all bytes 0–126. So, its current "next expected byte" is 127.
  - Host A sends Segment 1: 80 bytes of data, seq = 127 (starts with byte 127, covers bytes 127–206).
  - Host A sends Segment 2 (back-to-back): 40 bytes of data (covers bytes 207–246).
  - Ports in Segment 1: source (A) = 302, destination (B) = 80.

Problem 2.a

Sequence number, source port, and destination port in the second segment (from A to B)

a. Sequence number: This is always the byte position of the first byte in the segment. Segment 1 covers bytes 127–206 (80 bytes), so Segment 2 starts right after at byte 207. Thus, seq = 207.
b. Source port: This doesn't change—it's still Host A's port, 302.
c. Destination port: Still Host B's port, 80.

Correct answer: D) 207, 302, 80

Problem 2.b

If the first segment arrives before the second, what is the ack number, source port, and destination port in the acknowledgment of the first arriving segment (from B to A)?

- Acknowledgment number: Host B receives Segment 1 (bytes 127–206) in order (since it was expecting 127). It now has everything up to byte 206, so it acknowledges the next expected byte: 207.
- Source port: In the ACK (from B to A), ports swap—B's original destination port (80) becomes its source port: 80. Destination port: B sends back to A's original source port: 302.

Correct answer: C) 207, 80, 302

Problem 2.c

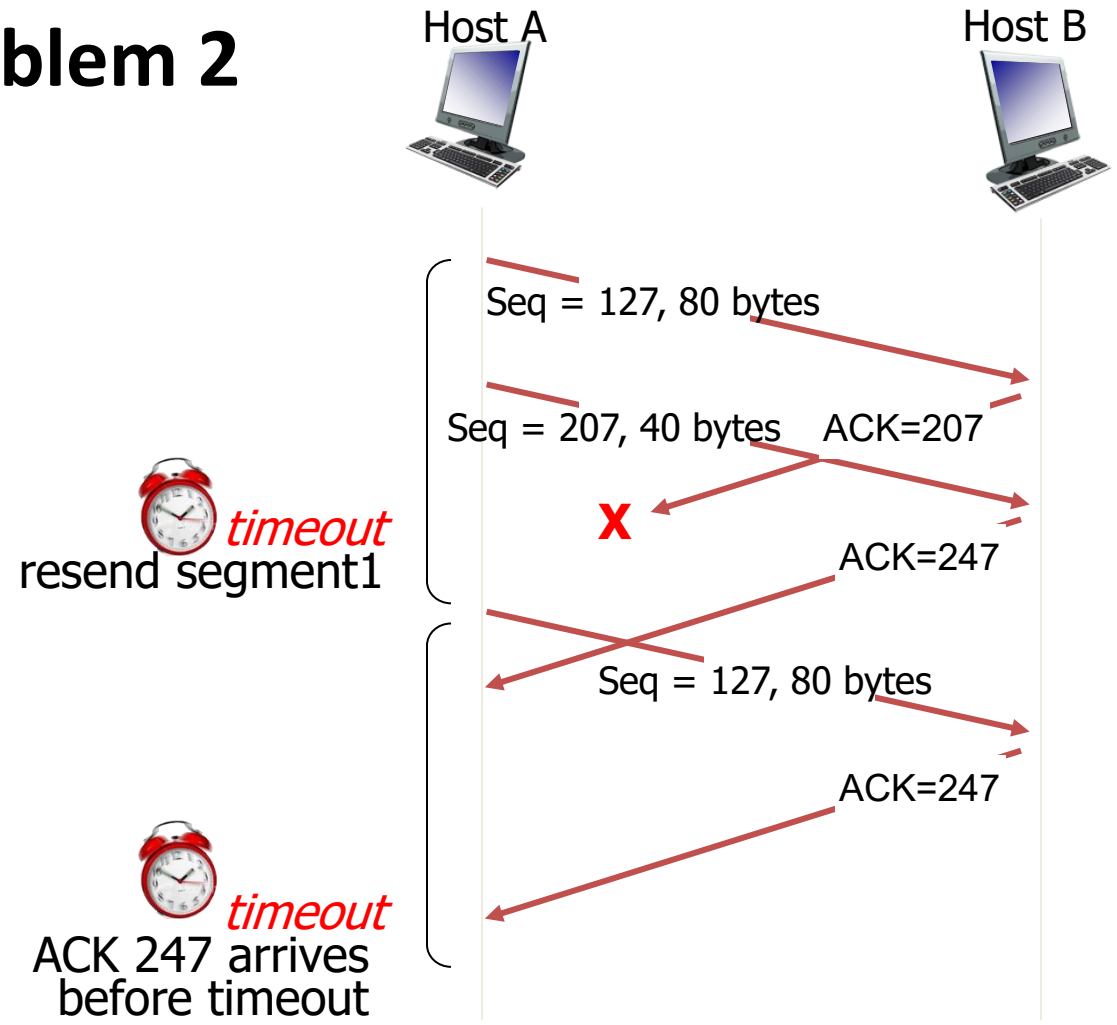
If the second segment arrives before the first, what is the acknowledgment number in the acknowledgment of the first arriving segment (from B to A)?

- Acknowledgment number: Host B receives Segment 2 first (bytes 207–246), but it's out of order—B is still expecting byte 127. TCP buffers out-of-order segments but doesn't advance its ack number until the gap is filled. So, B acknowledges its current next expected byte, which remains 127. (This prompts A to retransmit Segment 1 if needed.)

Correct answer: A) 127 This demonstrates TCP's reliable, ordered delivery: acks are cumulative

Problem 2

d)


Host A
Host B
Seq = 127, 80 bytes
Seq = 207, 40 bytes
ACK=207
X
timeout
resend segment1
ACK=247
Seq = 127, 80 bytes
ACK=247
timeout
ACK 247 arrives
before timeout

Timeout Interval for re-transmission

- In TCP, the retransmission timeout (RTO) interval—often called the TimeoutInterval—is adaptively computed to balance **responsiveness** and **stability** in the face of varying network conditions.
  - Given the values of EstimatedRTT (the smoothed estimate of the round-trip time) and DevRTT (the mean deviation of the RTT, capturing variability)

Problem 3

TimeoutInterval = EstimatedRTT + 4*DevRTT

estimated RTT

"safety margin"

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT

(typically, α = 0.125)

DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT|

(typically, β = 0.25)

The multiplier of 4 is empirically derived to set the timeout conservatively—about 4 standard deviations above the mean RTT

Problem 4

After obtaining the first sampleRTT 106 ms:

- EstimatedRTT = (1 – 0.125) · 100 + 0.125 · 106 = 0.875 · 100 + 13.25 = 100.75 ms

- DevRTT = (1 – 0.25) · 5 + 0.25 · |106 – 100.75| = 0.75 · 5 + 0.25 · 5.25 = 3.75 + 1.3125 = 5.0625 ms

- TimeoutInterval = 100.75 + 4* 5.0625 = 121 ms

Problem 4

After obtaining the second sampleRTT 120 ms:

- EstimatedRTT = 0.875 · 100.75 + 0.125 · 120 = 88.15625 + 15 = 103.15625 ms

- DevRTT = 0.75 · 5.0625 + 0.25 · |120 – 103.15625| = 3.796875 + 4.2109375 = 8.0078125 ms

- TimeoutInterval = 103.15625 + 4 · 8.0078125 = 103.15625 + 32.03125 = 135.1875 ms

Problem 4

After obtaining the third sampleRTT 140ms:

- EstimatedRTT = 0.875 · 103.15625 + 0.125 · 140 = 90.26171875 + 17.5 = 107.76171875 ms

- DevRTT = 0.75 · 8.0078125 + 0.25 · |140 – 107.76171875| = 6.005859375 + 8.0595703125 = 14.0654296875 ms

- TimeoutInterval = 107.76171875 + 4 · 14.0654296875 = 107.76171875 + 56.26171875 = 164.0234375 ms