TCP

Prof. Anja Feldmann, Ph.D.

(Based on slide deck of Computer Networking, 7th ed., Jim Kurose and Keith Ross.)

TCP: Overview

- ***Reliable, in-order byte stream***
  - No “message boundaries”
- ***Connection-oriented***
  - Handshaking *prior to* data exchange
- ***Flow controlled***
  - Sender *will not* overwhelm receiver
- ***Point-to-Point***
  - One sender, one receiver
- ***Full-duplex data channel***
  - *Bi-directional* data flow in same connection

RFCs

- *793,1122,1323, 2018, 2581*

Outline

- *Connection-oriented* transport: TCP
  - Quick refresher on TCP *Segment structure*
    - Sequence numbers & Acknowledgements
  - Reliable data transfer
  - Flow control
  - Connection management
- Congestion control
  - Principles
  - Mechanism

Outline

- *Connection-oriented* transport: TCP
  - Quick refresher on TCP *Segment structure*
    - Sequence numbers & Acknowledgements
  - Reliable data transfer
  - Flow control
  - Connection management
- Congestion control
  - Principles
  - Mechanism

TCP: Segment Structure

32 bits

<table>
<tr><td colspan="11">src. port</td><td>dst. port</td></tr>
<tr><td colspan="12">sequence number</td></tr>
<tr><td colspan="12">acknowledgment number</td></tr>
<tr><td>HL</td><td>*</td><td>N</td><td>C</td><td>E</td><td>U</td><td>A</td><td>P</td><td>R</td><td>S</td><td>F</td><td>receive window</td></tr>
<tr><td colspan="11">checksum</td><td>urgent pointer</td></tr>
<tr><td colspan="12">options (variable length)</td></tr>
<tr><td colspan="12">application data (variable length)</td></tr>
</table>

TCP: Sequence Numbers and ACKs

Sequence numbers

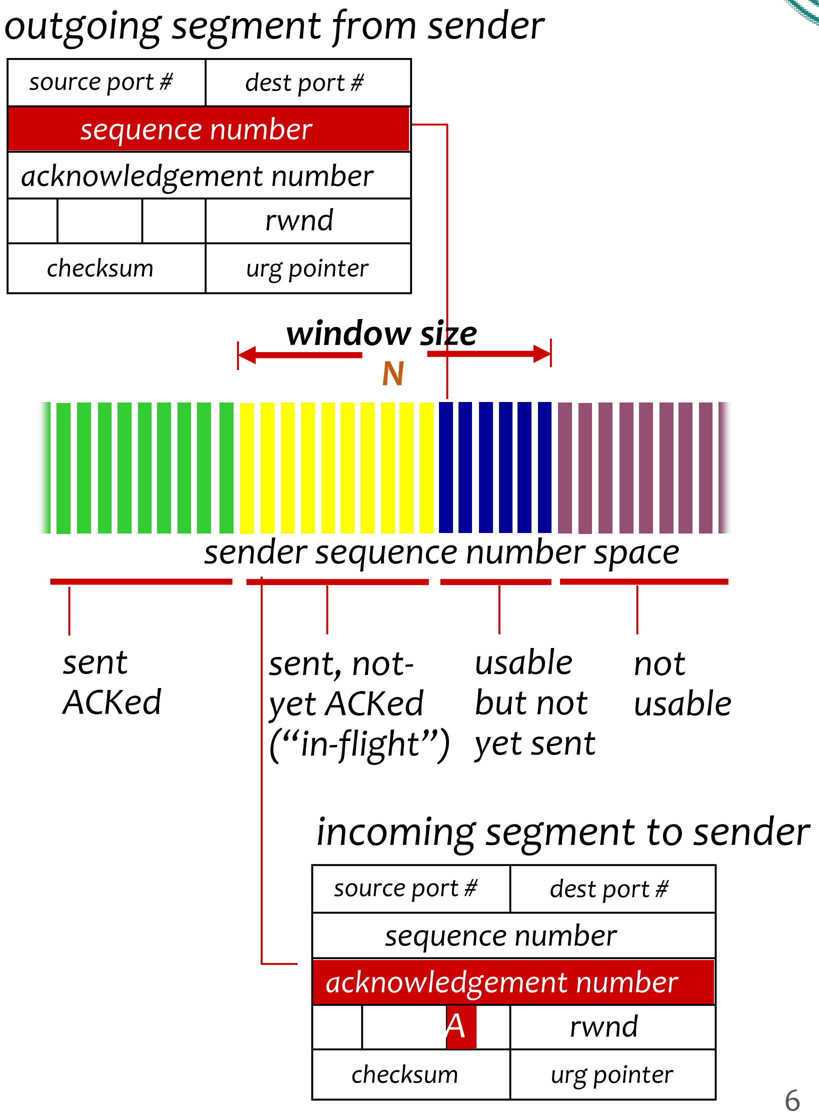
- Byte stream "*number*" of *first* byte in segment's data

Acknowledgements

- Sequence number of *next byte* expected from other side
- ***Cumulative*** ACK

How receiver handles out-of-order segments?

- TCP spec doesn't say; up to implementer!



outgoing segment from sender

source port #	*dest port #*
sequence number	
acknowledgement number	
	rwnd
checksum	*urg pointer*

window size

N

sender sequence number space

sent ACKed

sent, not-yet ACKed ("in-flight")

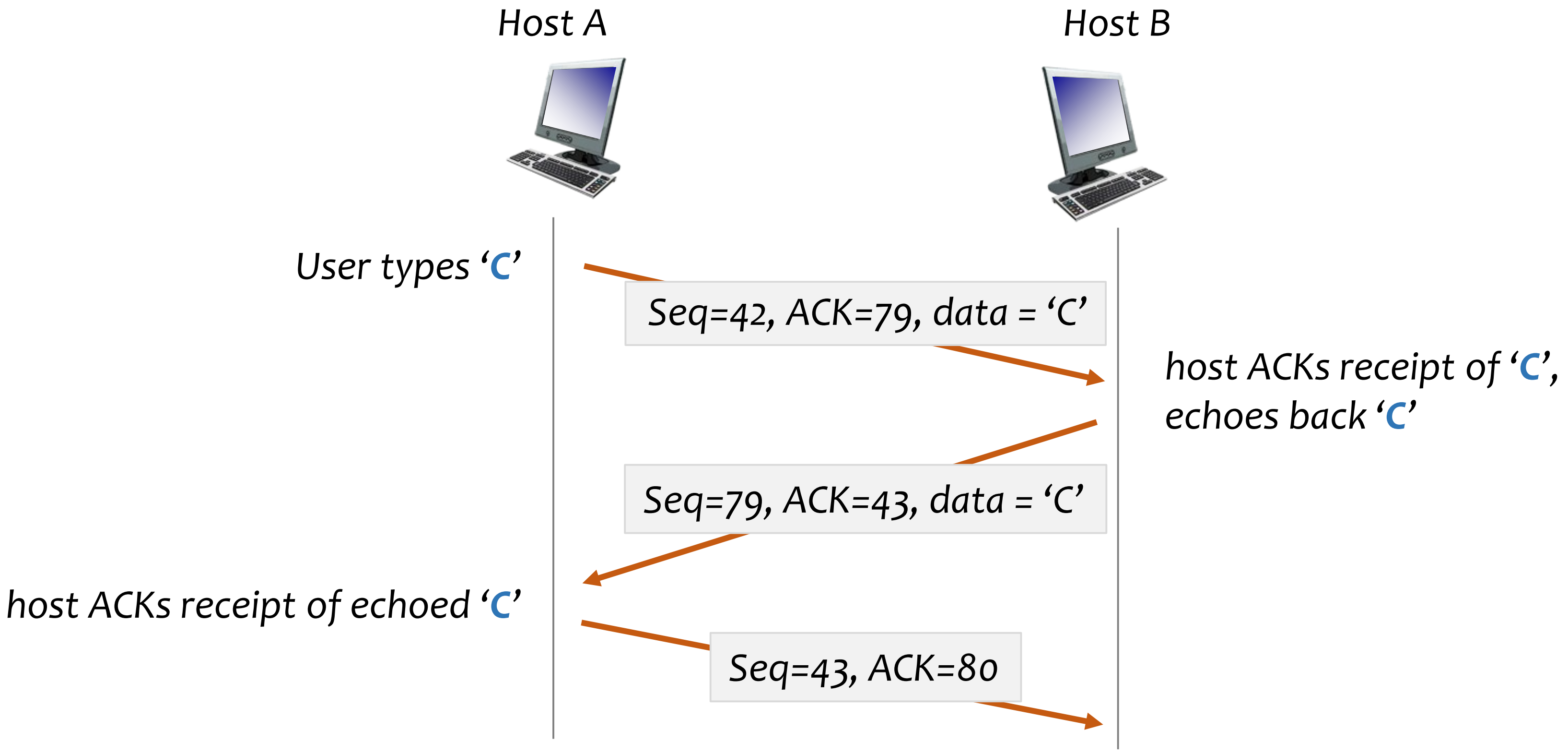
usable but not yet sent

not usable

incoming segment to sender

source port #	*dest port #*
sequence number	
acknowledgement number	
A	*rwnd*
checksum	*urg pointer*

TCP: Telnet Scenario



Host A

Host B

User types ‘C’

Seq=42, ACK=79, data = ‘C’

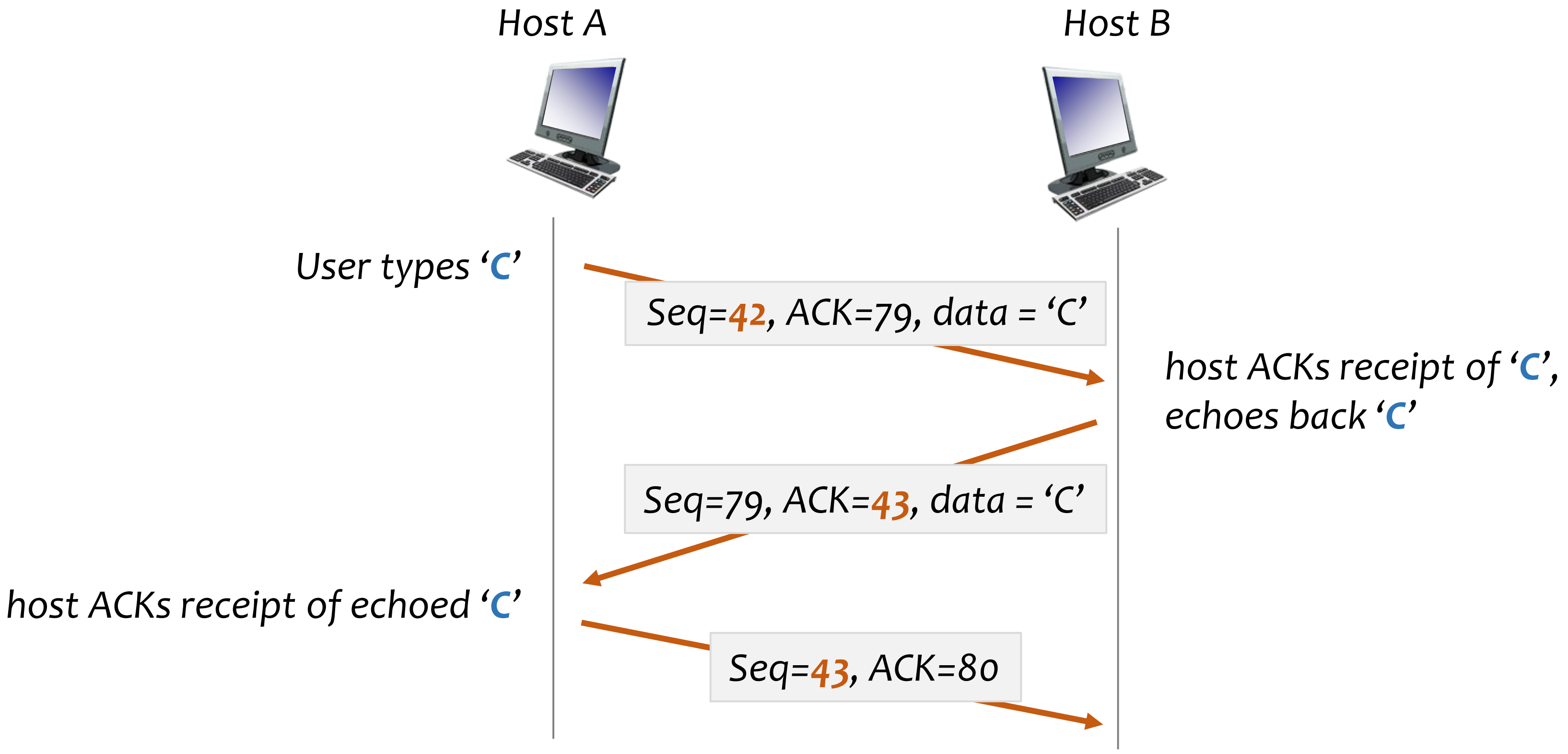
host ACKs receipt of ‘C’, echoes back ‘C’

Seq=79, ACK=43, data = ‘C’

host ACKs receipt of echoed ‘C’

Seq=43, ACK=80

TCP: Telnet Scenario



Host A

Host B

User types 'C'

Seq=42, ACK=79, data = 'C'

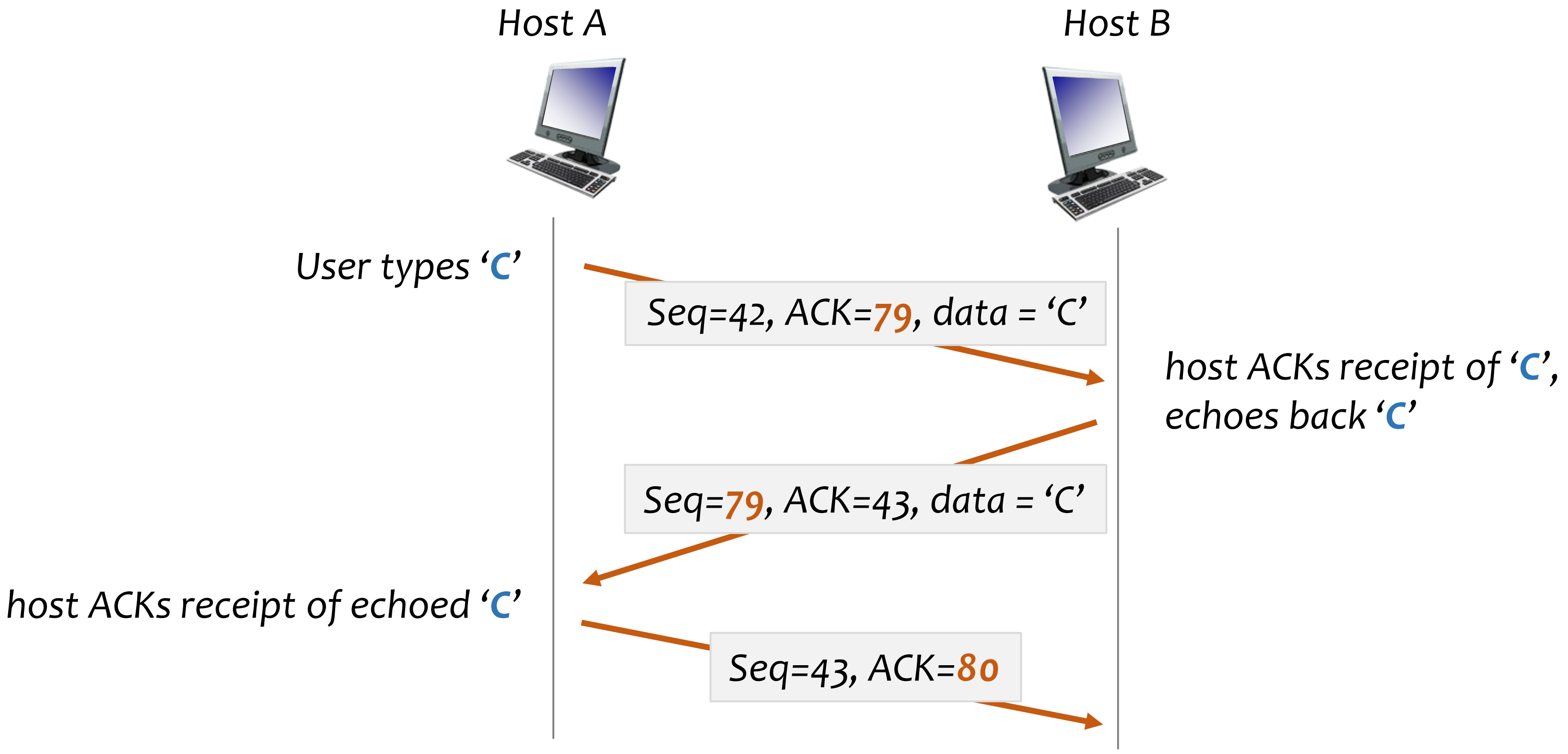
host ACKs receipt of 'C', echoes back 'C'

Seq=79, ACK=43, data = 'C'

host ACKs receipt of echoed 'C'

Seq=43, ACK=80

TCP: Telnet Scenario



Host A

Host B

User types ‘C’

Seq=42, ACK=79, data = ‘C’

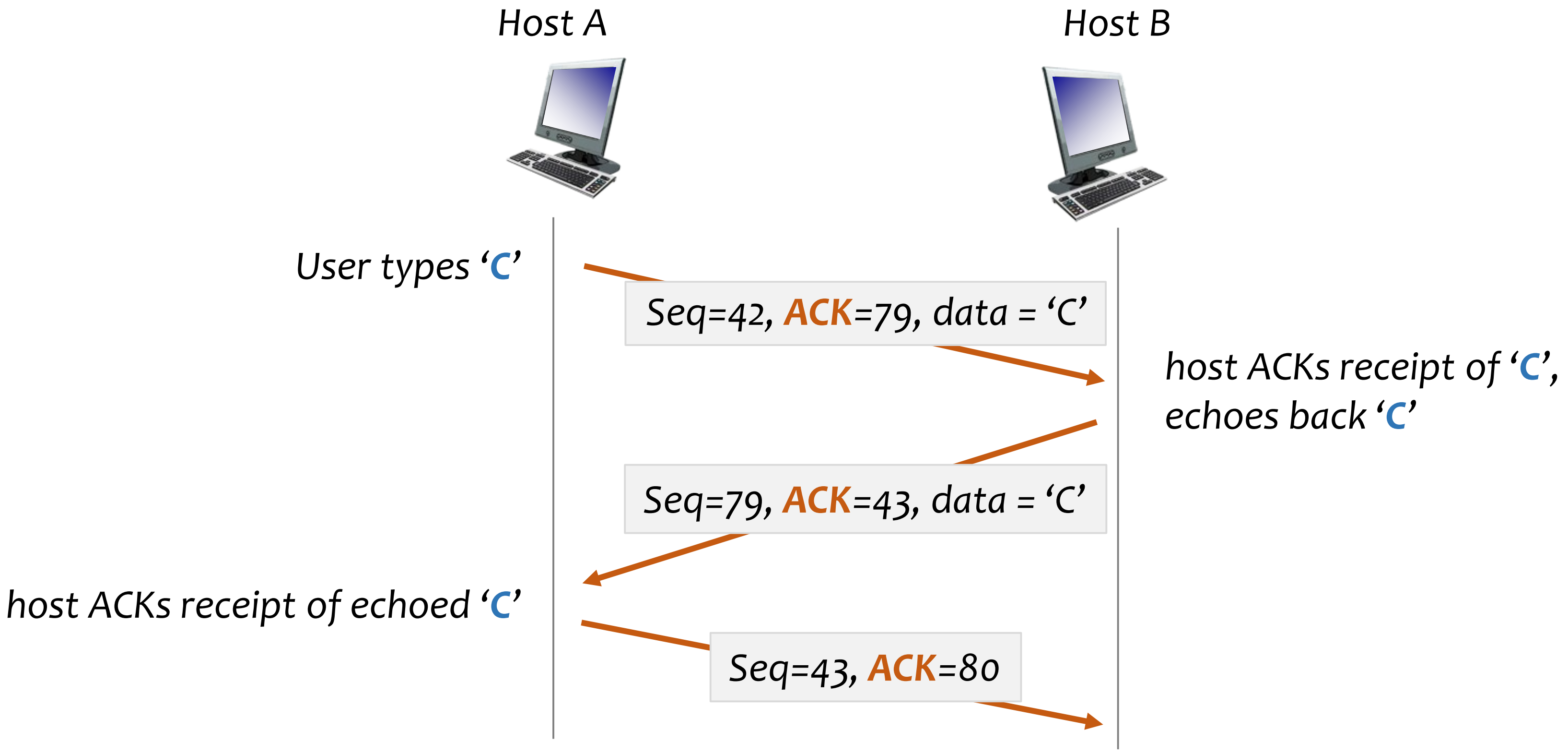
host ACKs receipt of ‘C’, echoes back ‘C’

Seq=79, ACK=43, data = ‘C’

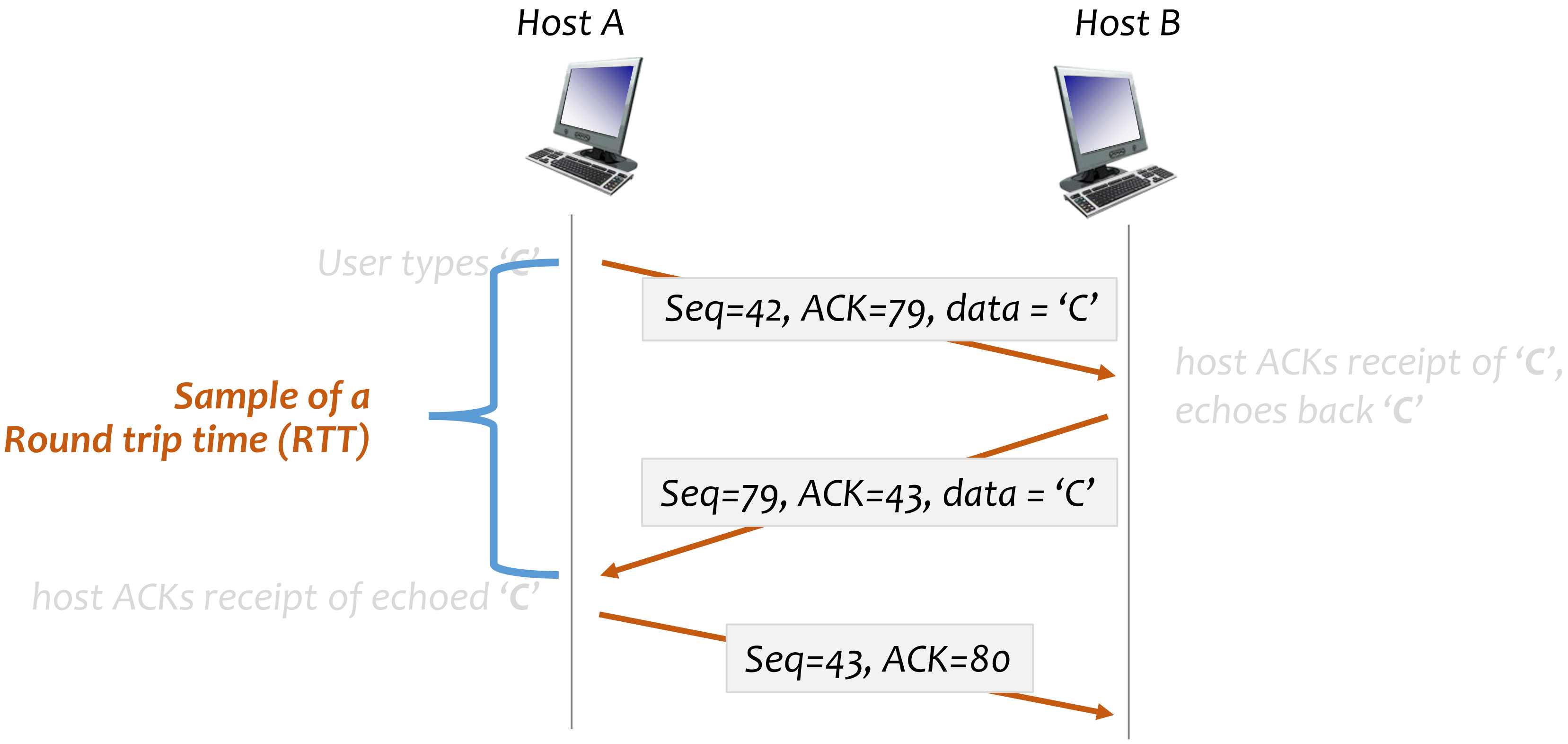
host ACKs receipt of echoed ‘C’

Seq=43, ACK=80

TCP: Telnet Scenario


Host A
Host B
User types ‘C’
Seq=42, ACK=79, data = ‘C’
host ACKs receipt of ‘C’, echoes back ‘C’
Seq=79, ACK=43, data = ‘C’
host ACKs receipt of echoed ‘C’
Seq=43, ACK=80

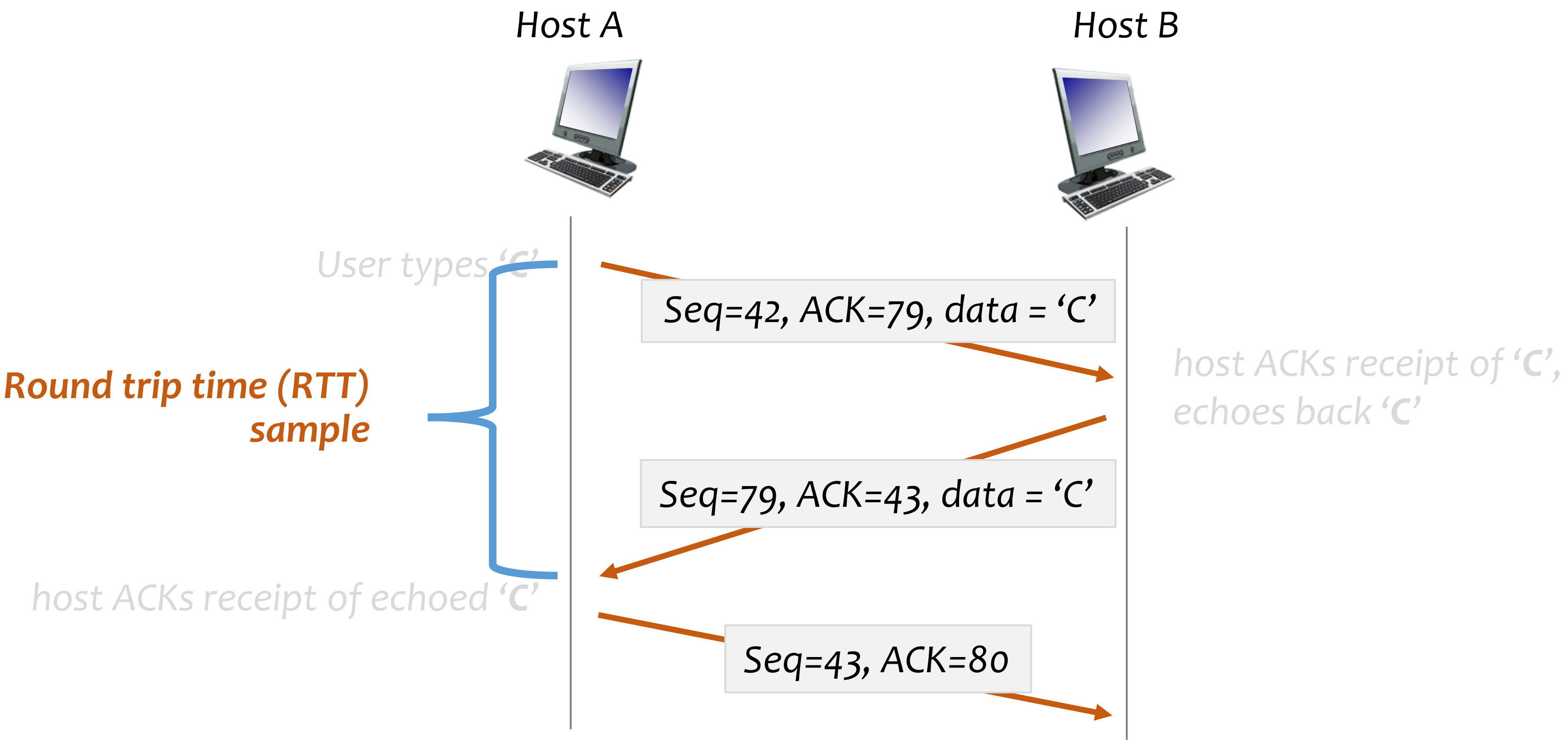
TCP: Round Trip Time (RTT)


Host A
Host B
User types 'C'
Seq=42, ACK=79, data = 'C'
host ACKs receipt of 'C', echoes back 'C'
Sample of a Round trip time (RTT)
Seq=79, ACK=43, data = 'C'
host ACKs receipt of echoed 'C'
Seq=43, ACK=80

TCP: Round Trip Time (RTT)

How long should the sender wait before ***retransmitting***?

- ***Timeout: Length of timer before the sender resends the segment***



Host A
Host B
User types 'C'
Seq=42, ACK=79, data = 'C'
host ACKs receipt of 'C', echoes back 'C'
Round trip time (RTT) sample
Seq=79, ACK=43, data = 'C'
host ACKs receipt of echoed 'C'
Seq=43, ACK=80

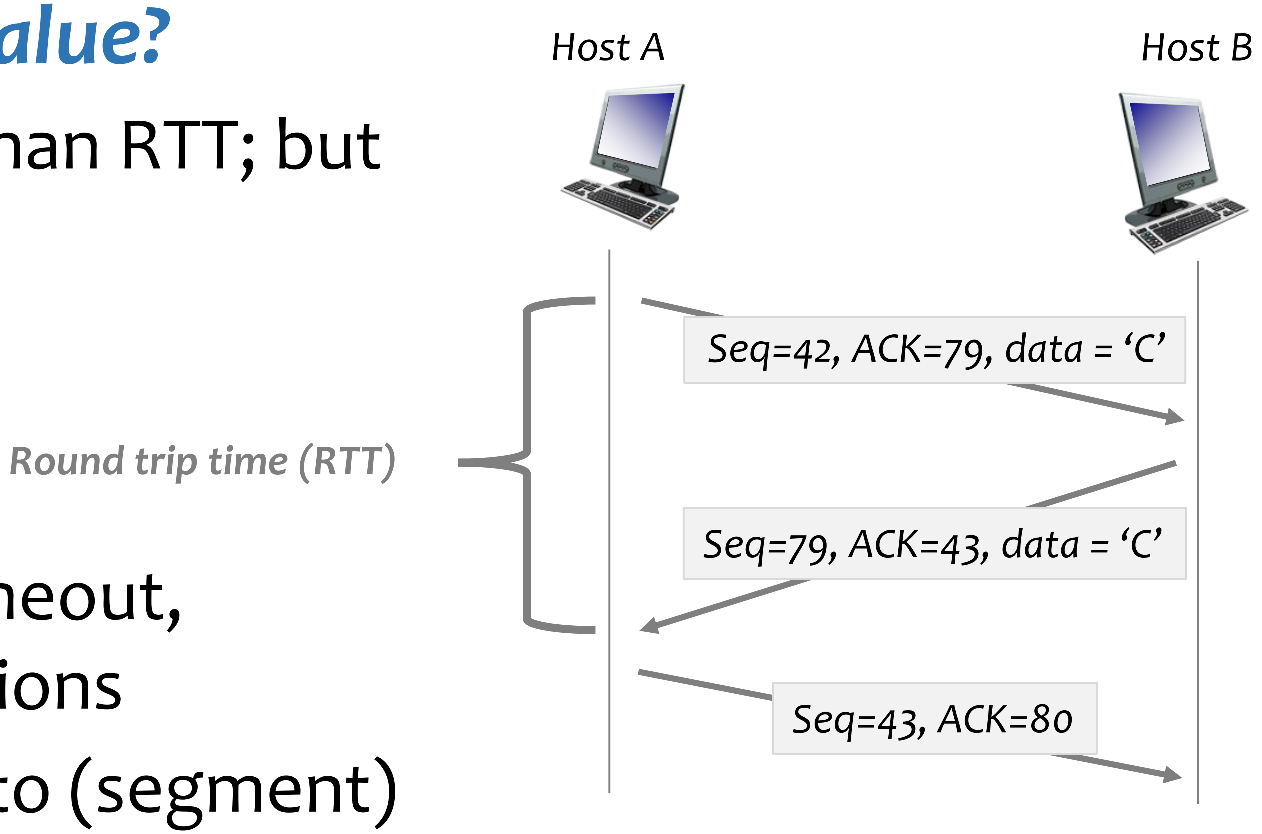
TCP: RTT & Timeout

How to set TCP timeout value?

- Set it to a value longer than RTT; but RTT varies!

Caveats?

- ***Too short***: *Premature* timeout, *unnecessary* retransmissions
- ***Too long***: *Slow* reaction to (segment) loss



TCP: RTT Estimation

How should we estimate RTT?

- ***SampleRTT***
  - Measured time from segment transmission until ACK receipt
  - *Ignore* retransmissions

*SampleRTT will vary; want **“smoother”** estimated RTT*

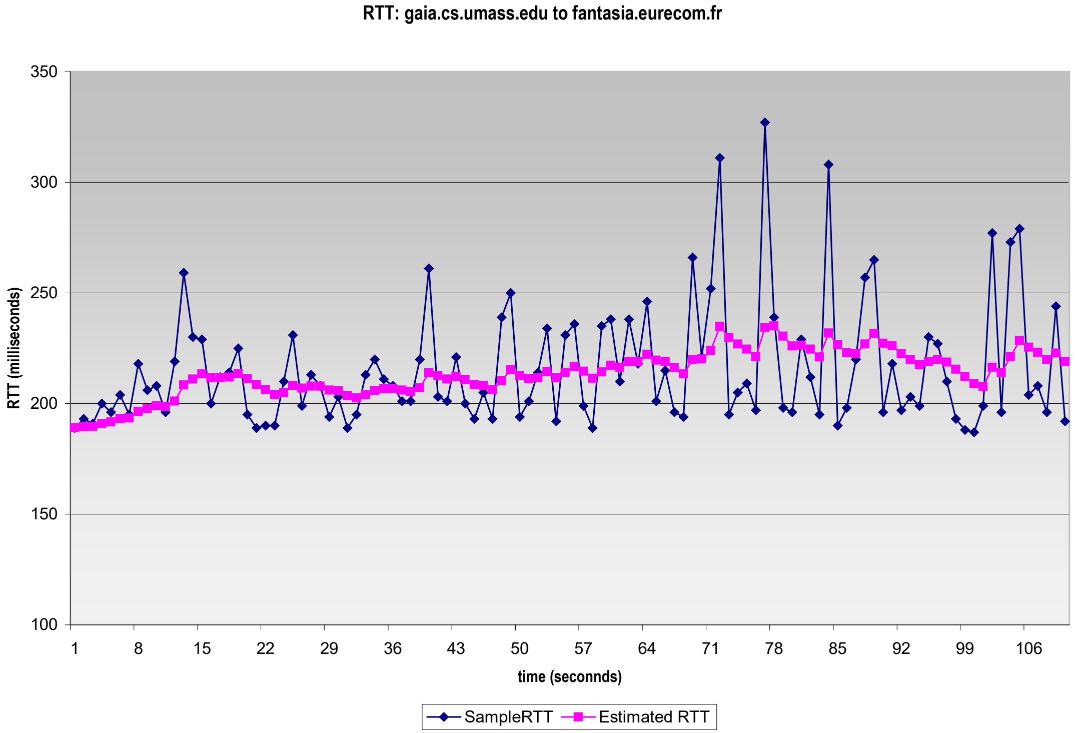
- *Average* several *recent* measurements (*i.e., not just current SampleRTT)*

TCP: RTT Estimation

$$\textit{EstimatedRTT} = (1-\alpha) * \textit{EstimatedRTT} + \alpha * \textit{SampleRTT}$$

Exponential weighted moving average

- Influence of past sample decreases *exponentially fast*
- Typical value: $\alpha = 0.125$



TCP: Timeout

Timeout interval: *EstimatedRTT* plus *"safety margin"*

- Large variation in *EstimatedRTT* → larger safety margin
- Estimate *SampleRTT* ***deviation (DevRTT)*** from EstimatedRTT:

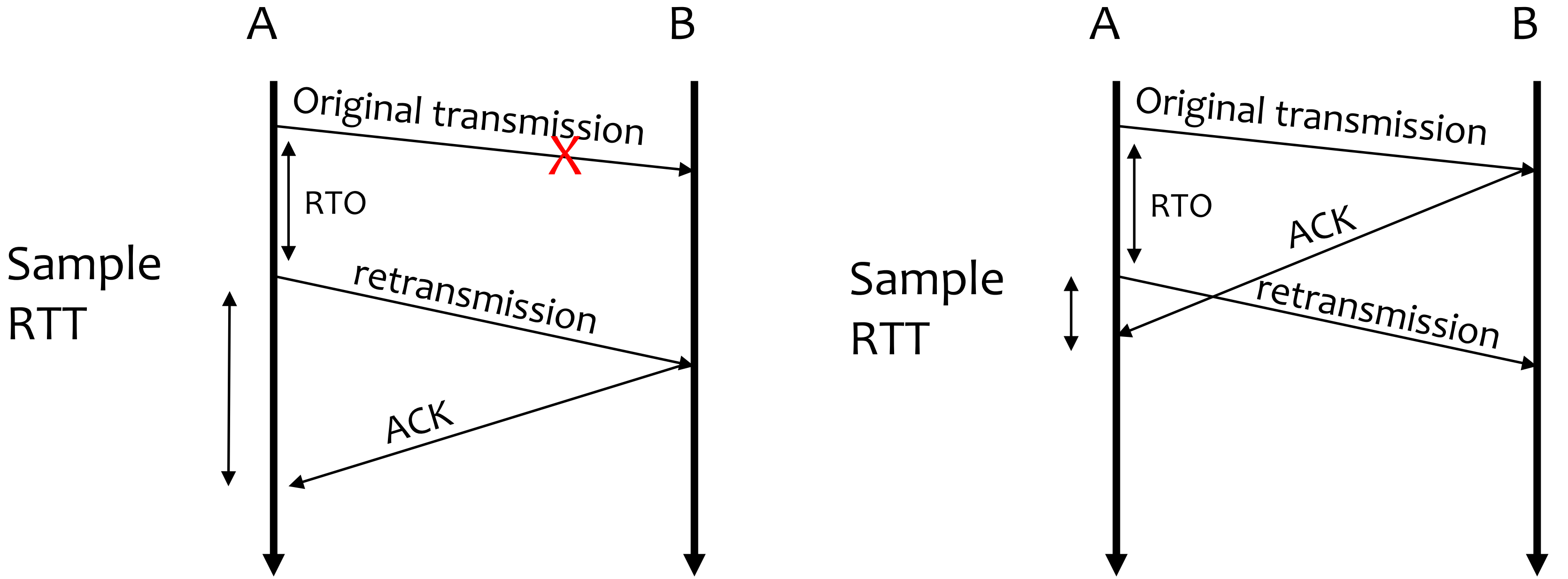
$$DevRTT = (1-\beta) * DevRTT + \beta * |SampleRTT - EstimatedRTT|$$

(*typically*, β = 0.25)

$$TimeoutInterval = EstimatedRTT + 4*DevRTT$$

("4*DevRTT": *Safety margin*)

Retransmission Ambiguity



Karn´s RTT Estimator

- If a segment has been retransmitted:
  - ***Do not count RTT*** sample on ACKs for this segment
- Keep backed off ***time-out*** for next packet
- Reuse RTT estimate only after one successful transmission

Outline

- *Connection-oriented* transport: TCP
  - Quick refresher on TCP *Segment structure*
    - Sequence numbers & Acknowledgements
  - Reliable data transfer
  - Flow control
  - Connection management
- Congestion control
  - Principles
  - Mechanism

TCP: Reliable Data Transfer (RDT)

- TCP creates RDT service on top of IP’ s unreliable service
  - Pipelined segments
  - Cumulative ACKs
  - Single retransmission timer

- Retransmissions triggered by:
  - Timeout events
  - Duplicate ACKs

Let’s initially consider a simplified TCP sender:

- Ignore duplicate ACKs
- Ignore flow control, congestion control

TCP Sender Events:

Data rcvd from app:

- Create segment with sequence number
- Sequence number is byte-stream number of first data byte in segment
- Start timer if not already running
  - Think of timer as for oldest un-Ack'd segment
  - Expiration interval: ***TimeOutInterval***

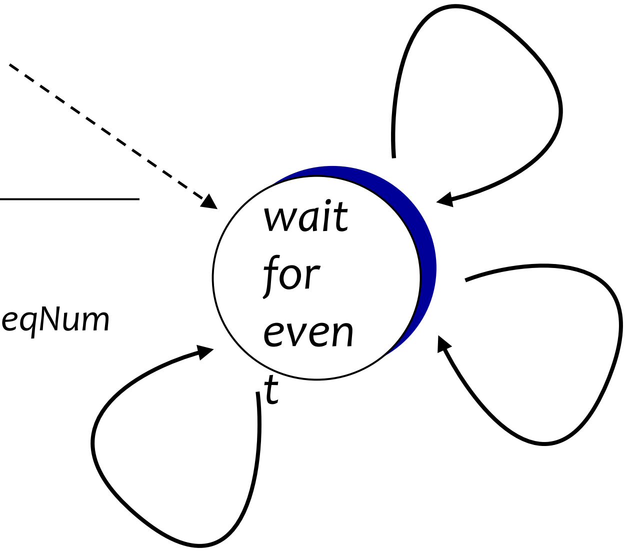
Timeout:

- Retransmit segment that caused timeout
- Restart timer

ACK rcvd.:

- If ACK acknowledges previously un-ACK'd segments
  - Update what is known to be ACK'd
  - Start timer if there are still un-ACK'd segments

TCP Sender (simplified)



Λ

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

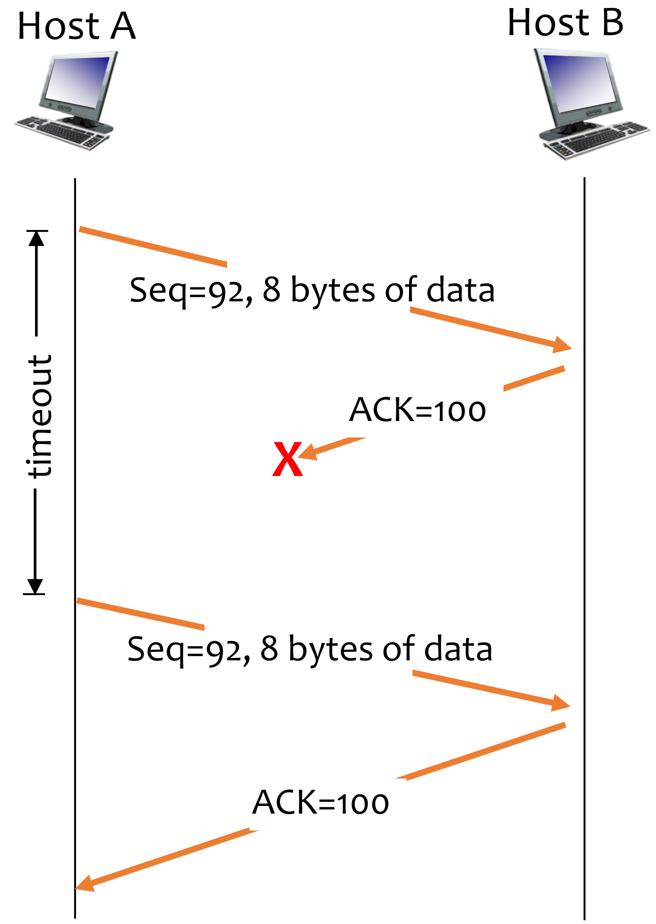
timeout

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

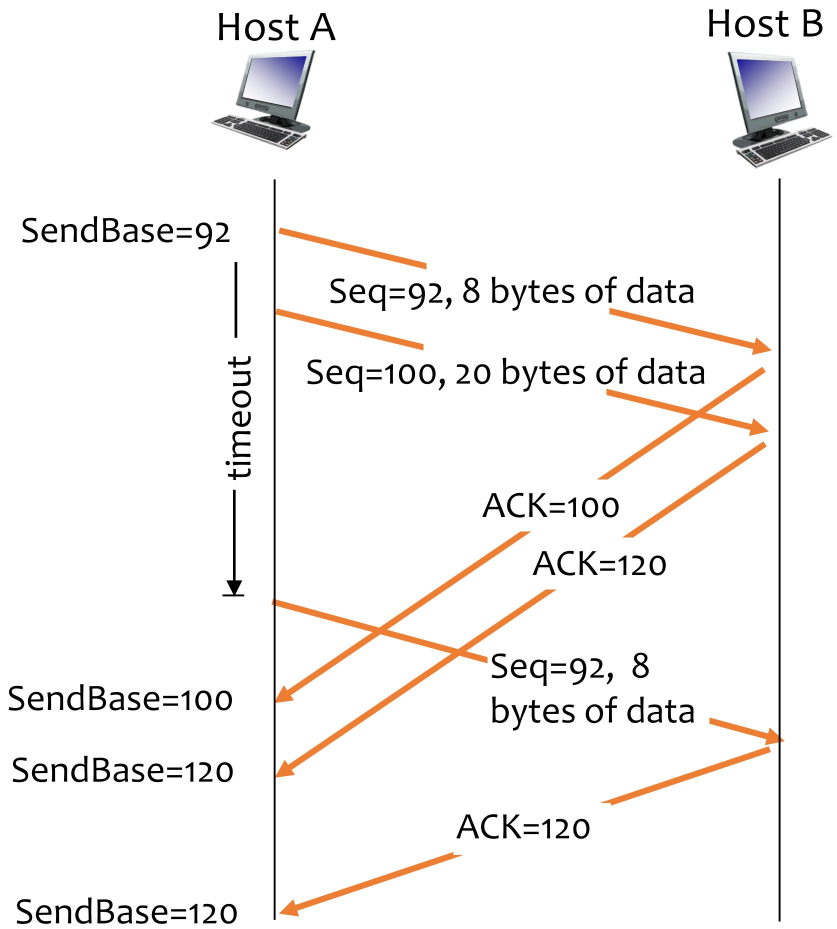
ACK received, with ACK field value y

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

TCP: Retransmission Scenarios

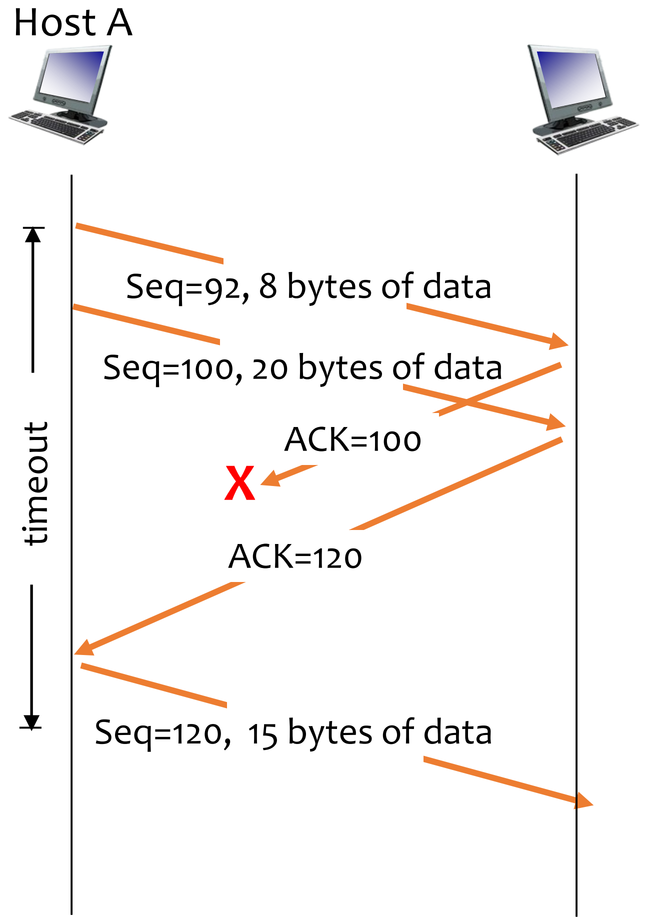


Lost ACK scenario



Premature timeout

TCP: Retransmission Scenarios



Cumulative ACK

TCP ACK Generation [RFC 1122, RFC 2581]

Event at receiver	*TCP receiver action*
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq. # . Gap detected	Immediately send *duplicate ACK*, indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediate send ACK, provided that segment starts at lower end of gap

TCP Fast Retransmit

- Time-out period often relatively long:
  - Long delay before resending lost packet
- Detect lost segments via duplicate ACKs.
  - Sender often sends many segments back-to-back
  - If segment is lost, there will likely be many duplicate ACKs.

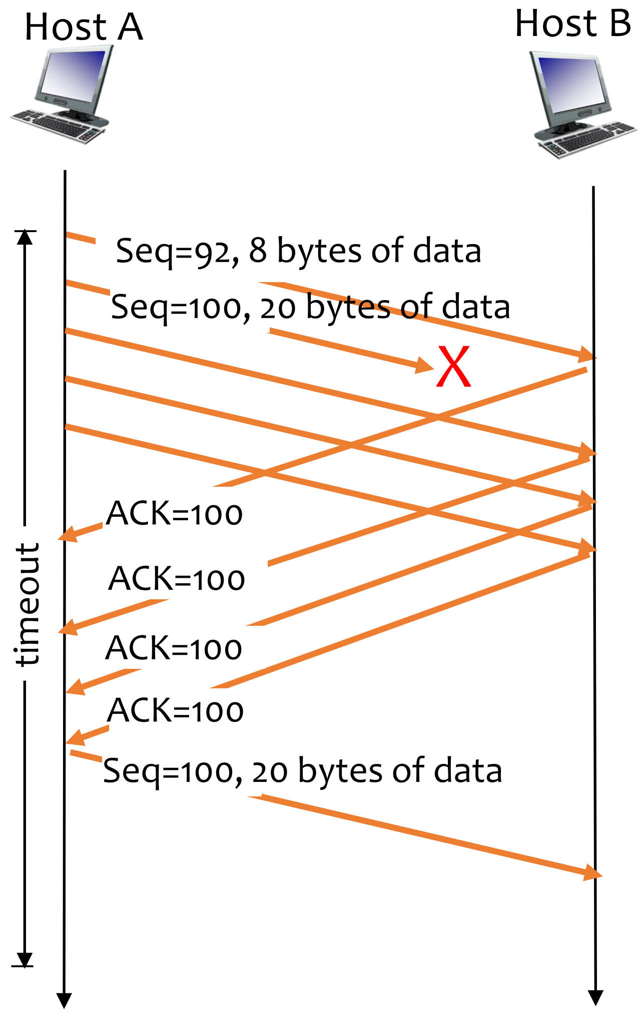
TCP fast retransmit

If sender receives 3 ACKs for same data ("triple duplicate ACKs"), resend unacked segment with smallest seq #

- Likely that unacked segment lost, so don' t wait for timeout

TCP Fast Retransmit

Fast retransmit after sender receipt of triple duplicate ACK



Outline

- *Connection-oriented* transport: TCP
  - Quick refresher on TCP *Segment structure*
    - Sequence numbers & Acknowledgements
  - Reliable data transfer
  - Up next: Flow control
  - Up next: Connection management
- Congestion control