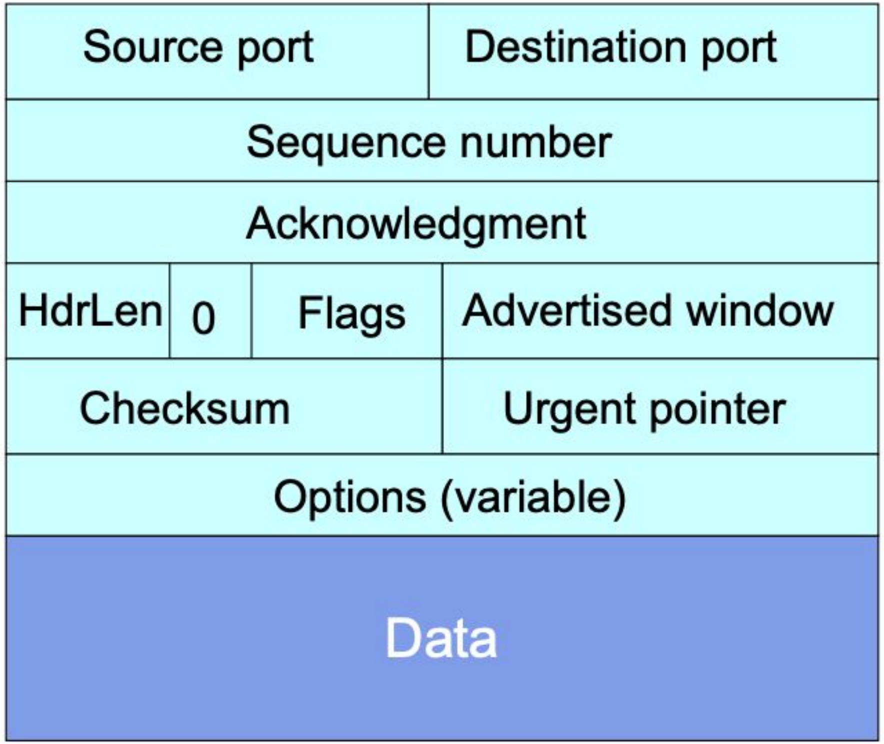
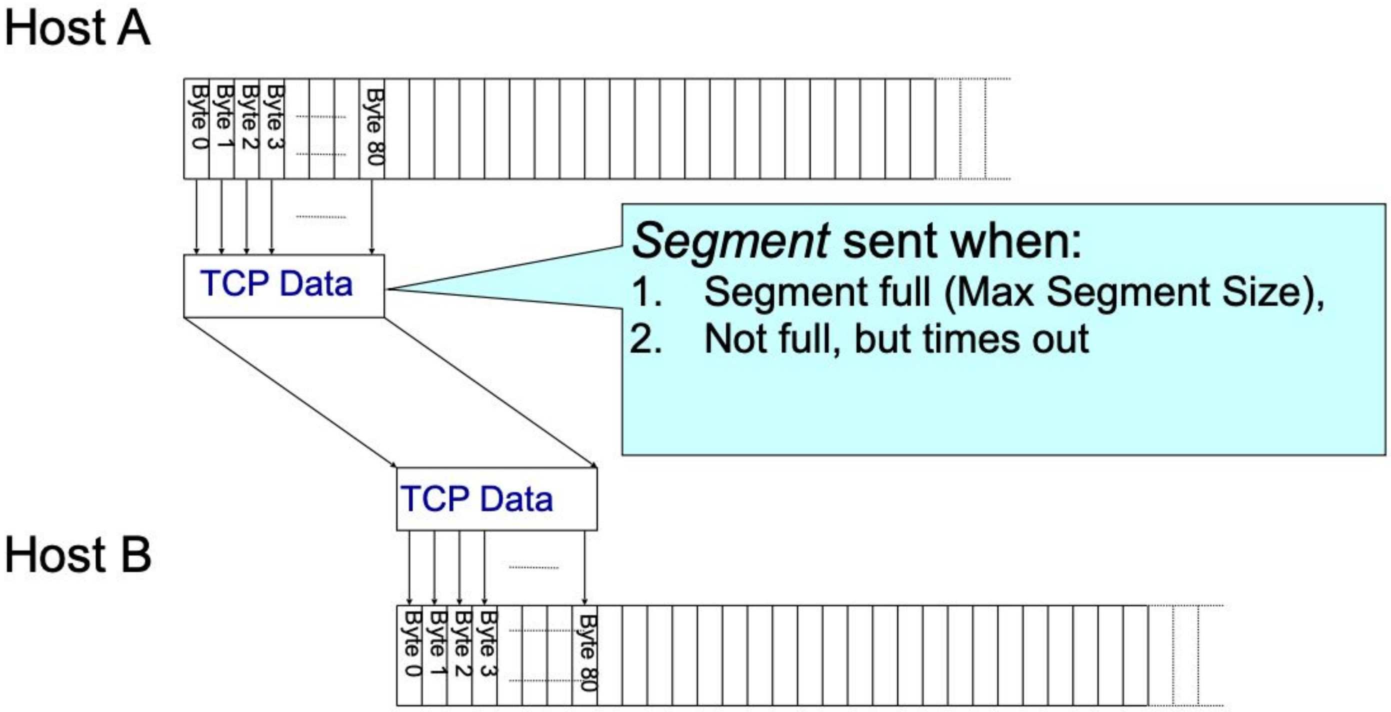
TCP Header


Source port
Destination port
Sequence number
Acknowledgment
HdrLen
0
Flags
Advertised window
Checksum
Urgent pointer
Options (variable)
Data

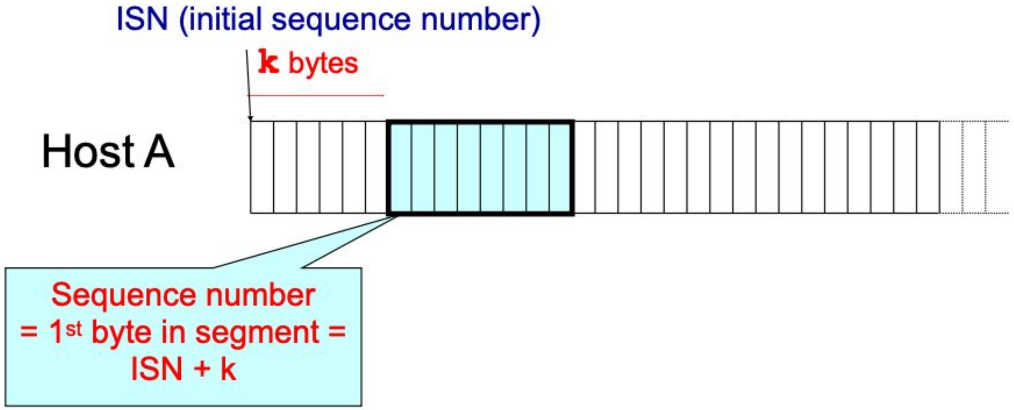
… Provided Using TCP "Segments"



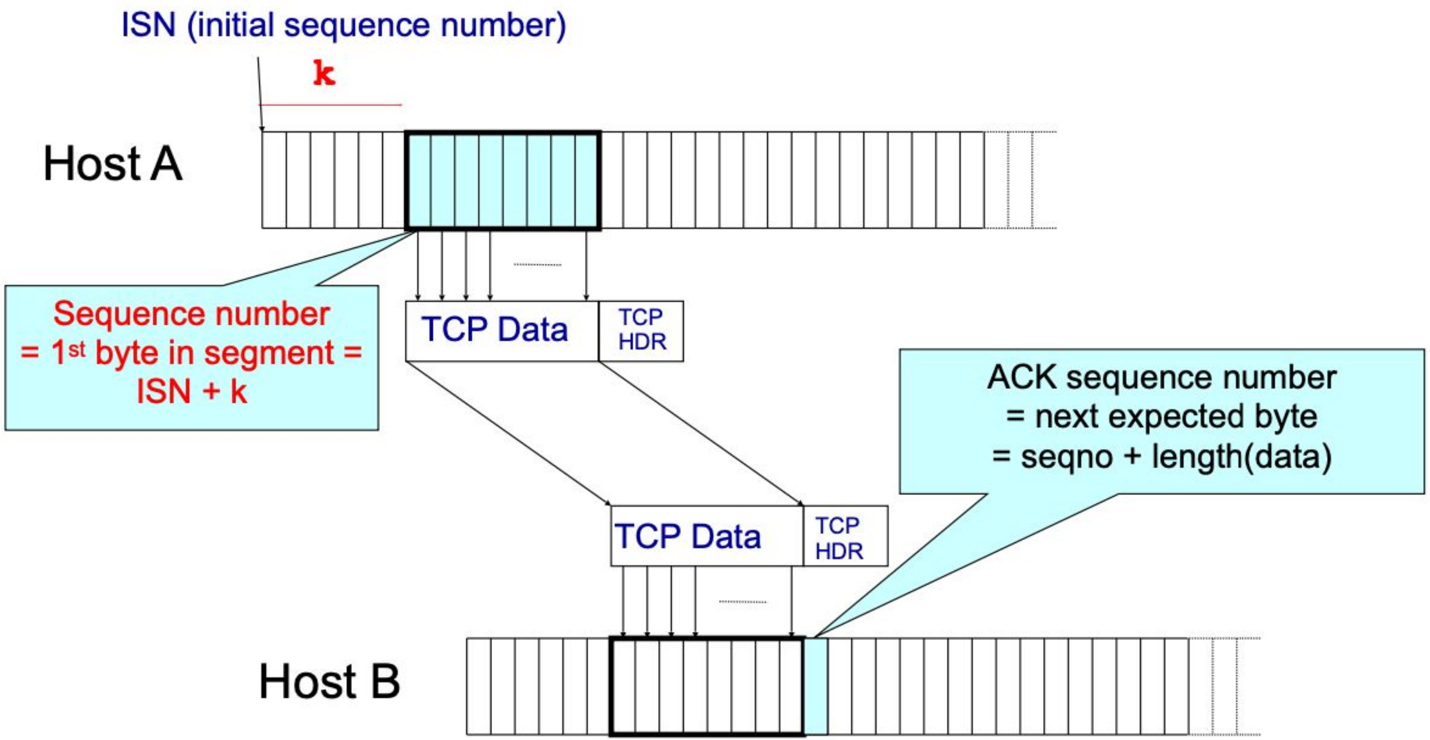
TCP Segment

- IP packet
  - No bigger than Maximum Transmission Unit (MTU)
  - E.g., up to 1500 bytes with Ethernet
- TCP packet
  - IP packet with a TCP header and data inside
  - TCP header >= 20 bytes long
- TCP segment
  - No more than Maximum Segment Size (MSS) bytes
  - E.g., up to 1460 consecutive bytes from the stream
  - MSS = MTU – (IP header) – (TCP header)

Sequence Numbers



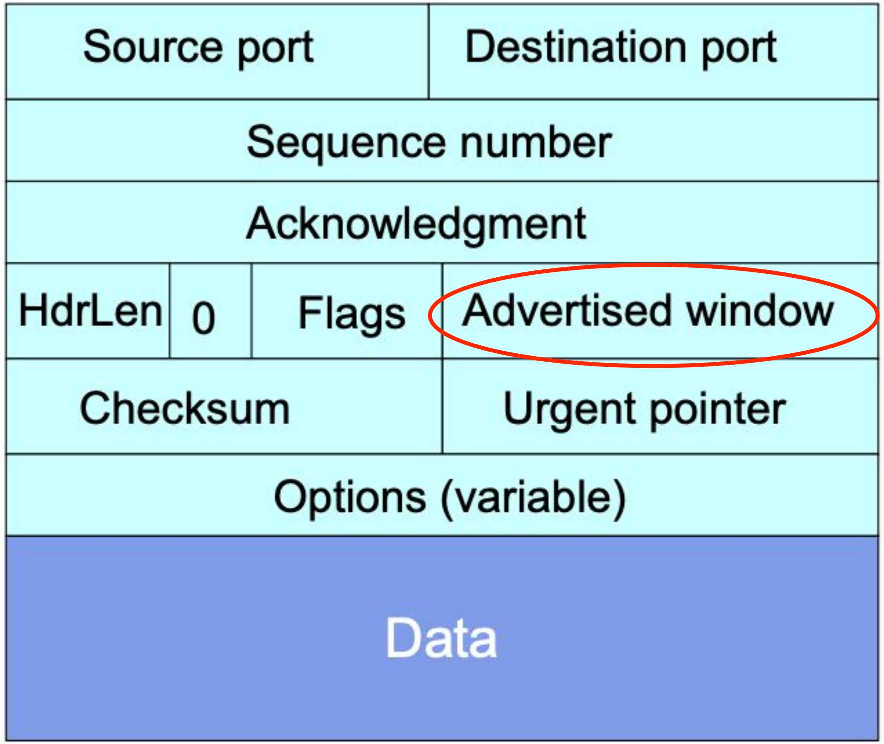
Sequence Numbers



ACKing and Sequence Numbers

- Sender sends packet
  - Data starts with sequence number X
  - Packet contains B bytes
    - X, X+1, X+2, ...X+B-1
- Upon receipt of packet, receiver sends an ACK
  - If all data prior to X already received:
    - ACK acknowledges X+B (because that is next expected byte)
  - If highest contiguous byte received is smaller value Y
    - ACK acknowledges Y+1
    - Even if this has been ACKed before

TCP Header


Source port
Destination port
Sequence number
Acknowledgment
HdrLen
0
Flags
Advertised window
Checksum
Urgent pointer
Options (variable)
Data

Sliding Window Flow Control

- Advertised Window: W
  - Can send W bytes beyond the next expected byte
- Receiver uses W to prevent sender from overflowing buffer
- Limits number of bytes sender can have in flight

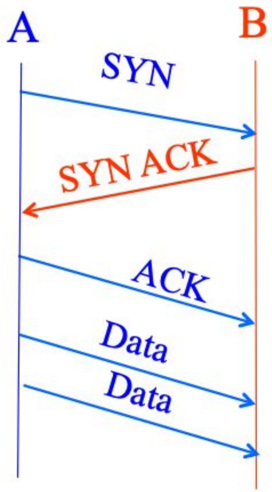
Advertised Window Limits Rate

Sender can send no faster than W/RTT bytes/sec

Receiver only advertises more space when it has consumed old arriving data

In original TCP design, that was the sole protocol mechanism controlling sender's rate

Establishing a TCP Connection

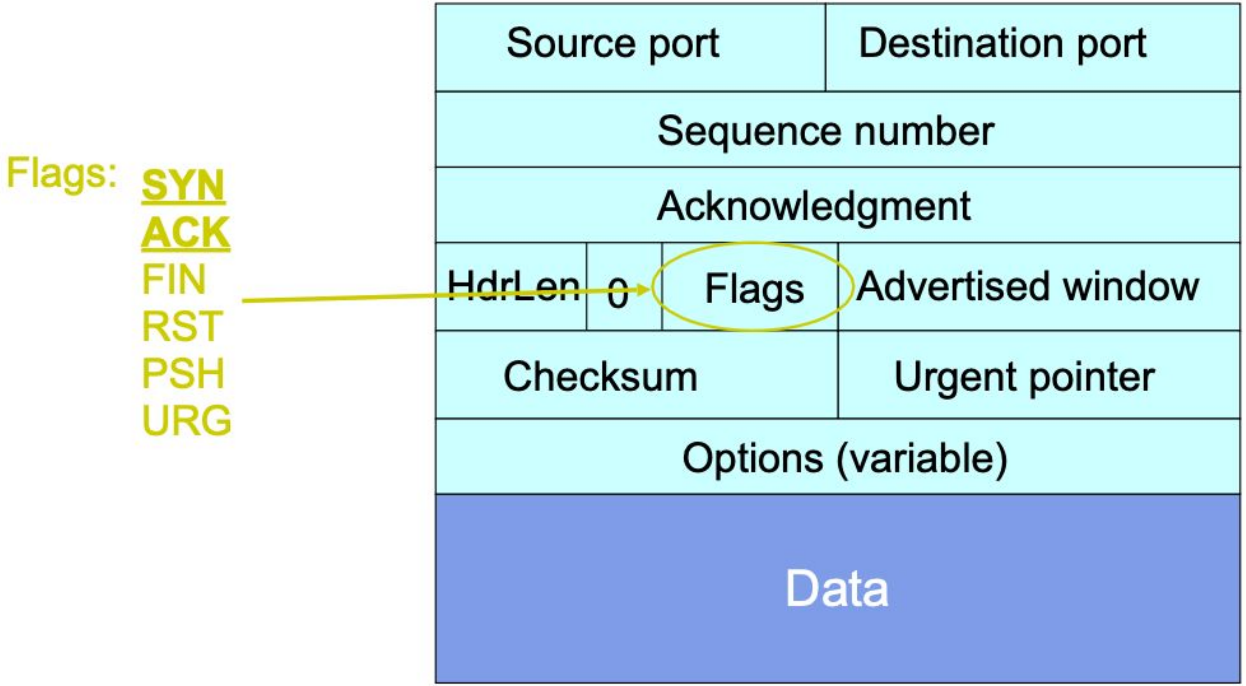


Each host tells its ISN to the other host.

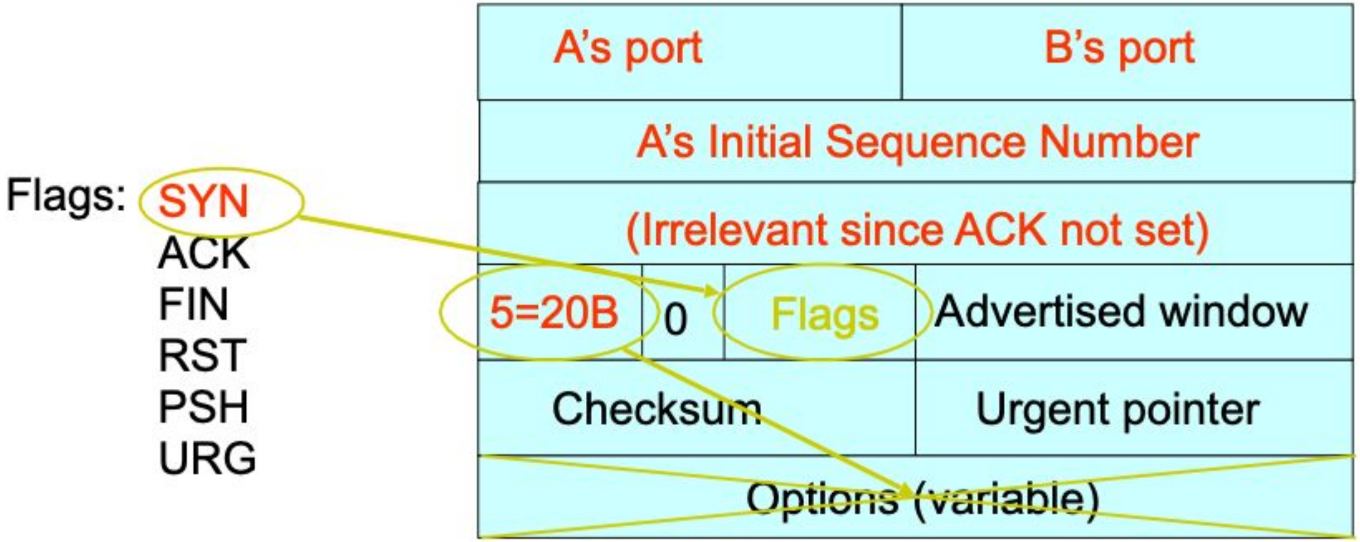
Three-way handshake to establish connection

- Host A sends a **SYN** (open; "synchronize sequence numbers")
- Host B returns a SYN acknowledgment (**SYN ACK**)
- Host A sends an **ACK** to acknowledge the SYN ACK

TCP Header

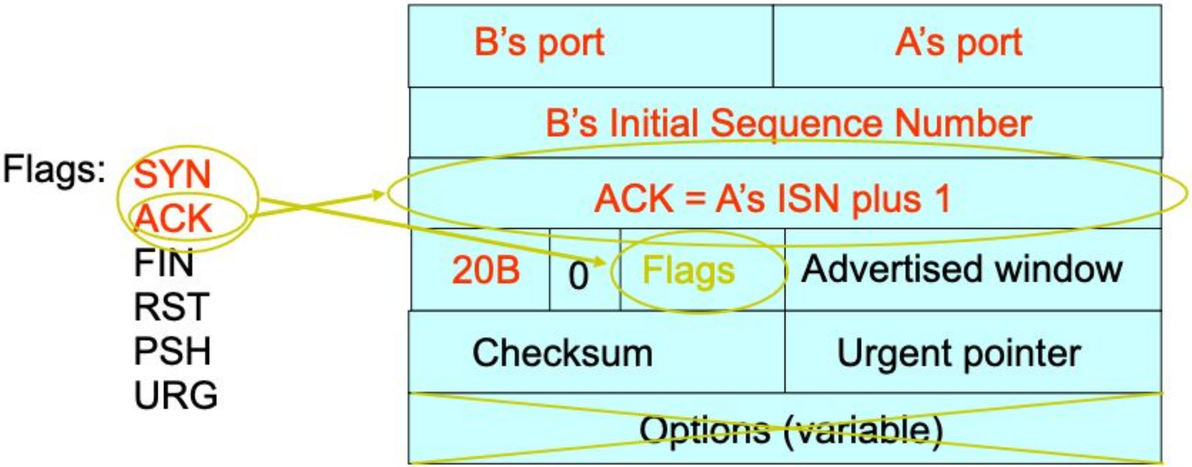

Flags:
SYN
ACK
FIN
RST
PSH
URG
Source port
Destination port
Sequence number
Acknowledgment
HdrLen
0
Flags
Advertised window
Checksum
Urgent pointer
Options (variable)
Data

Handshake step 1: A's initial SYN packet



A tells B it wants to open a connection…

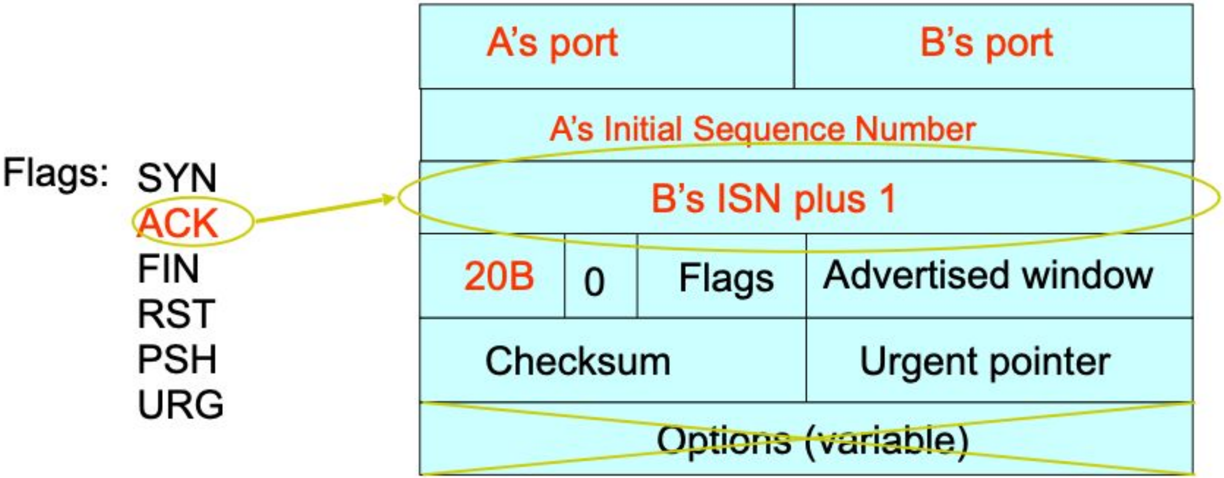
Handshake step 2: B’s SYN-ACK packet



B tells A it accepts, and is ready to hear the next byte…

… upon receiving this packet, A can start sending data

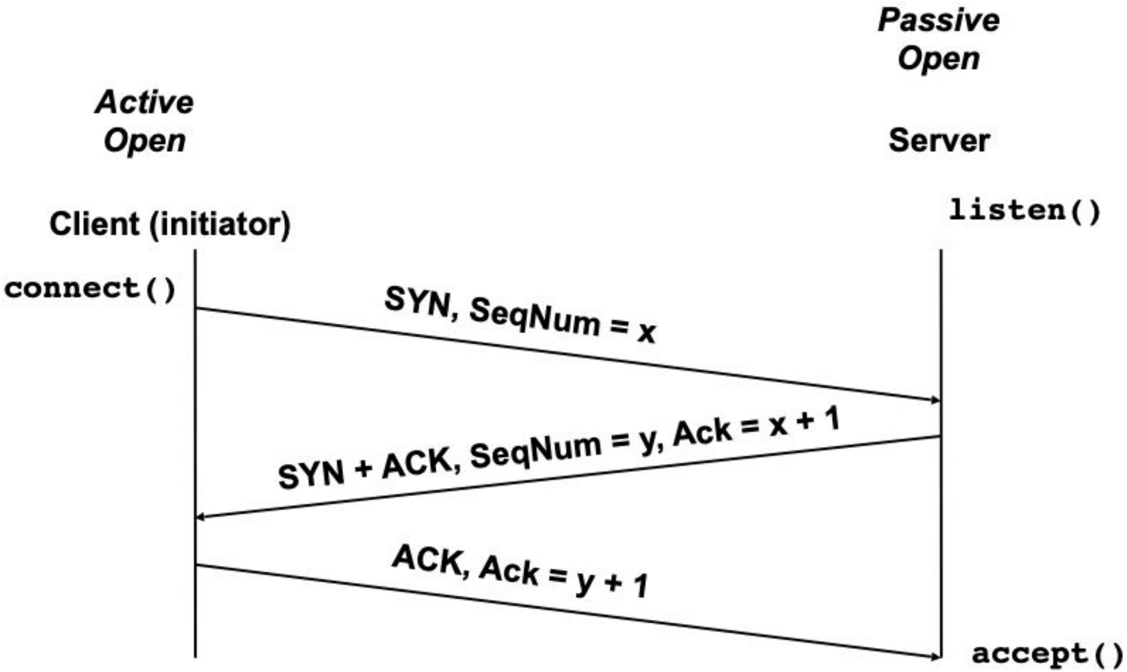
Handshake step 3: A's ACK of the SYN-ACK packet



A tells B it's likewise okay to start sending

… upon receiving this packet, B can start sending data

Timing Diagram: 3-Way Handshaking

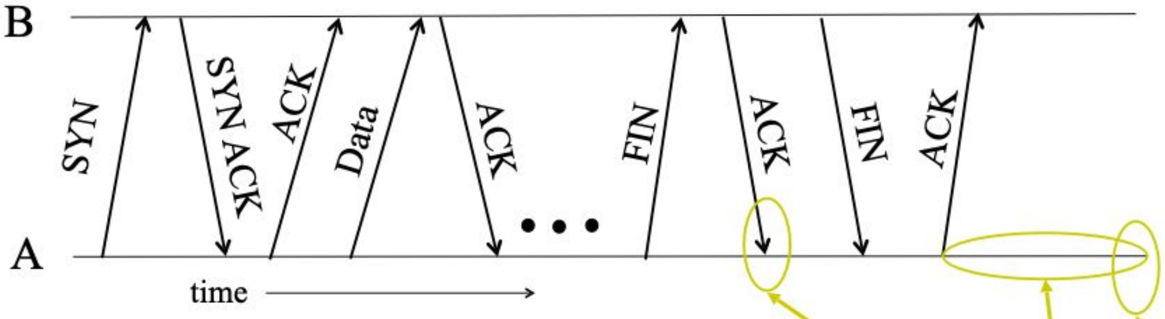


What if the SYN Packet Gets Lost?

- Suppose the SYN packet gets lost
  - Packet is lost inside the network, or:
  - Server discards the packet (e.g., listen queue is full)
- Eventually, no SYN-ACK arrives
  - Sender sets a timer and waits for the SYN-ACK
  - ... and retransmits the SYN if needed
- How should the TCP sender set the timer?
  - Sender has no idea how far away the receiver is
  - Hard to guess a reasonable length of time to wait
  - SHOULD (RFCs 1122 & 2988) use default of 3 seconds
    - Other implementations instead use 6 seconds

TCP Connection Teardown

Normal Termination, One Side At A Time



Finish (**FIN**) to close and receive remaining bytes

- **FIN** occupies one octet in the sequence space

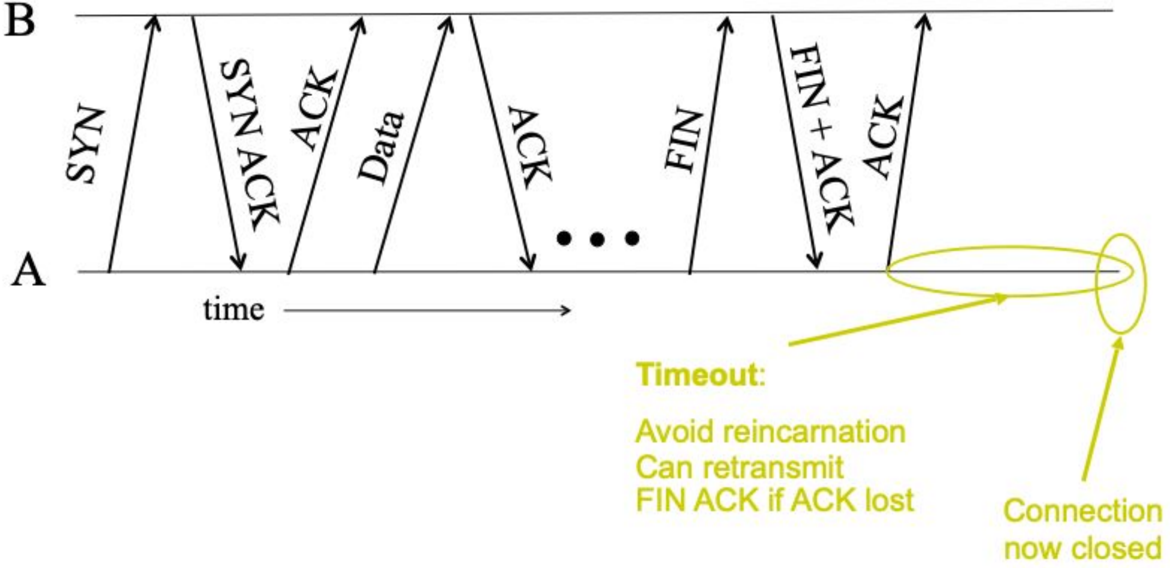
Other host ack's the octet to confirm

Closes A's side of the connection, but not B's

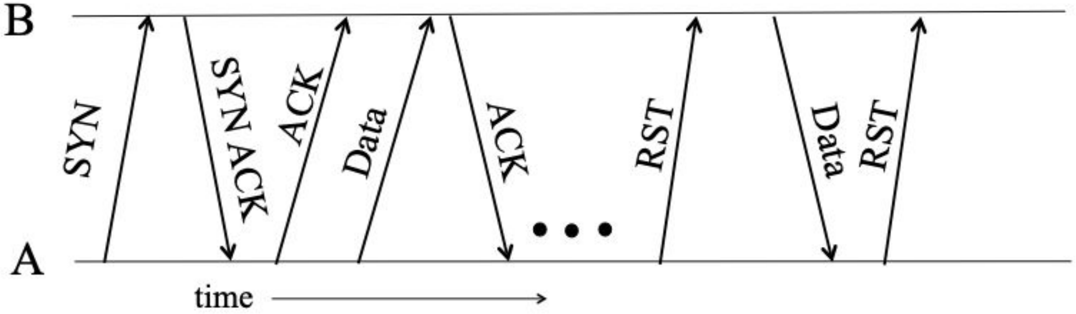
- Until B likewise sends a **FIN**
- Which A then acks

Normal Termination, Both Together

Same as before, but B sets **FIN** with their ack of A's **FIN**



Abrupt Termination



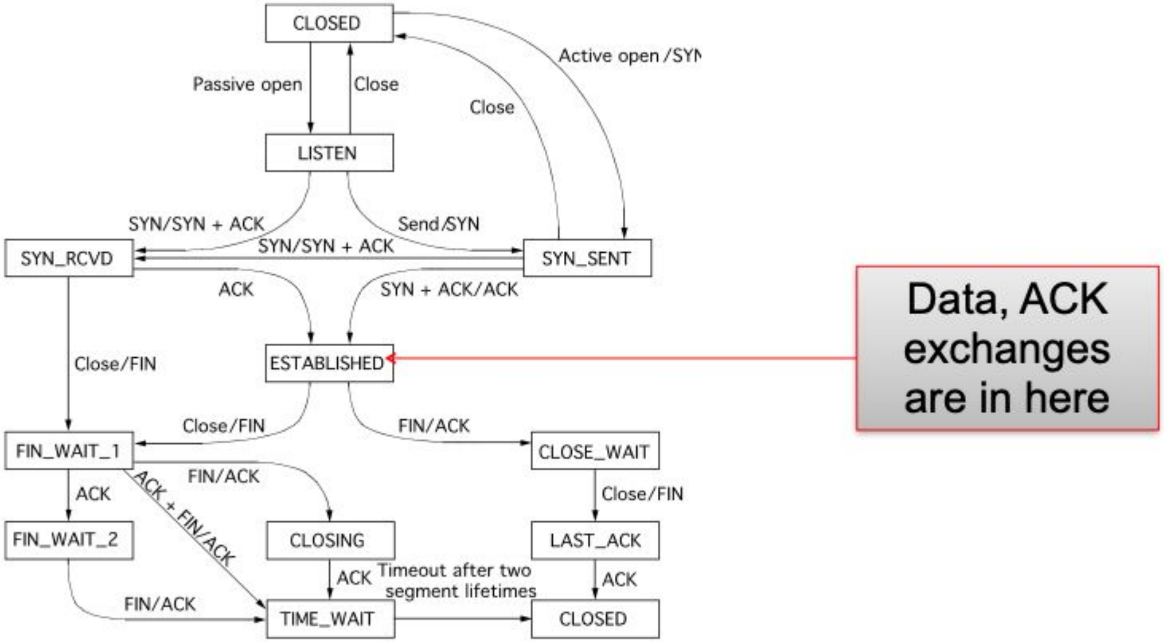
A sends a RESET (**RST**) to B

- E.g., because app. process on A crashed

That's it

- B does not ack the **RST**
- Thus, **RST** is not delivered reliably
- And: any data in flight is lost
- But: if B sends anything more, will elicit another **RST**

TCP State Transitions



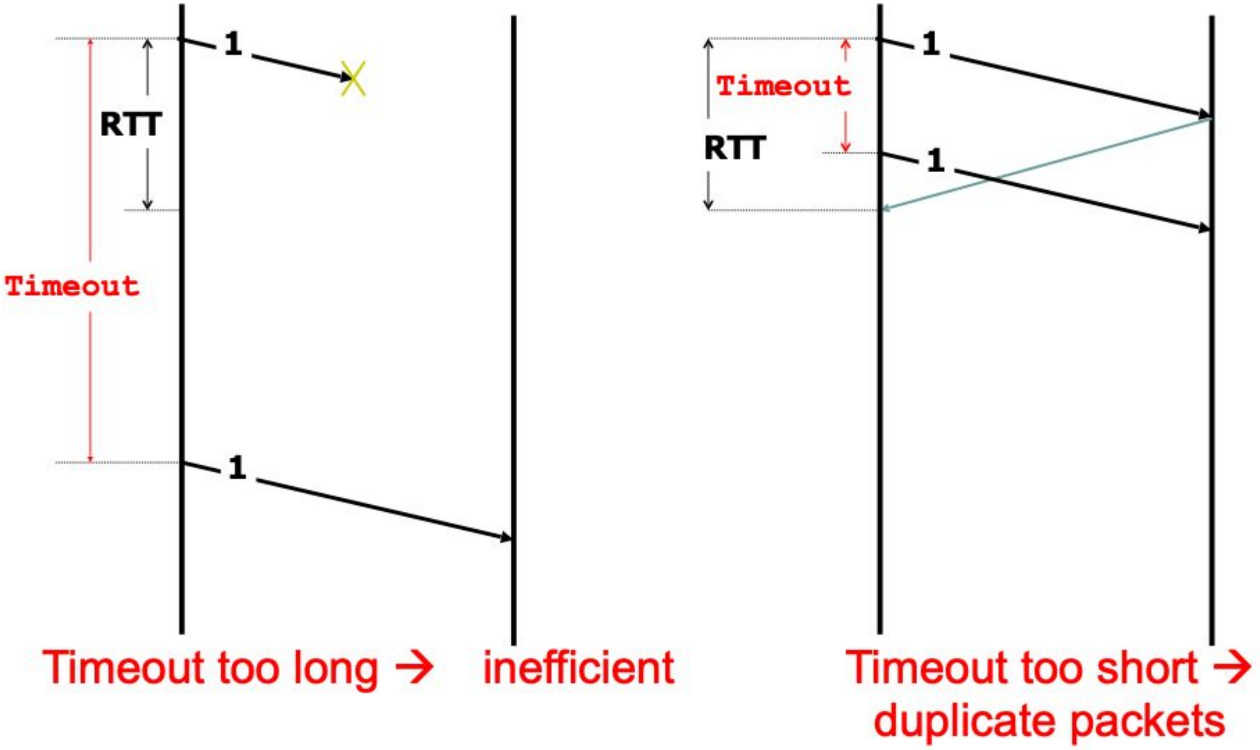
Reliability: TCP Retransmissions

- Reliability requires retransmitting lost data
- Involves setting timer and retransmitting on timeout
- TCP resets timer whenever new data is ACKed
    - Retx of packet containing “next byte” when timer goes off

Example

- Arriving ACK expects 100
- Sender sends packets 100, 200, 300, 400, 500
  - Timer set for 100
- Arriving ACK expects 300
  - Timer set for 300
- Timer goes off
  - Packet 300 is resent
- Arriving ACK expects 600
  - Packet 600 sent
  - Timer set for 600

Setting the Timeout Value



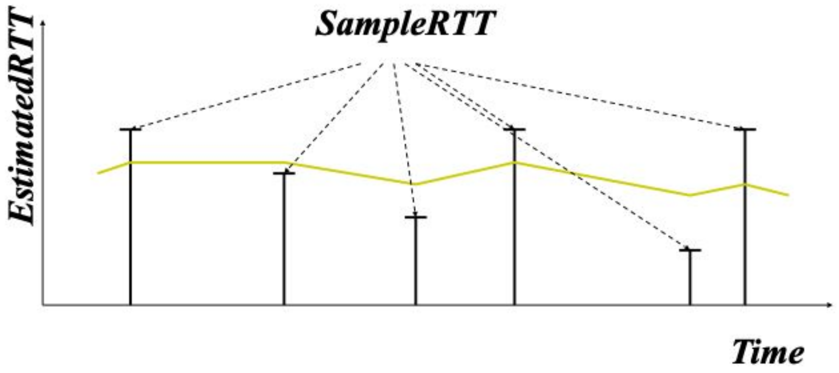
RTT Estimation

Use exponential averaging of RTT samples

$$SampleRTT = AckRcvdTime - SendPacketTime$$

$$EstimatedRTT = \alpha \times EstimatedRTT + (1 - \alpha) \times SampleRTT$$

$$0 < \alpha \leq 1$$



Exponential Averaging Example

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

Assume RTT is constant → *SampleRTT* = RTT