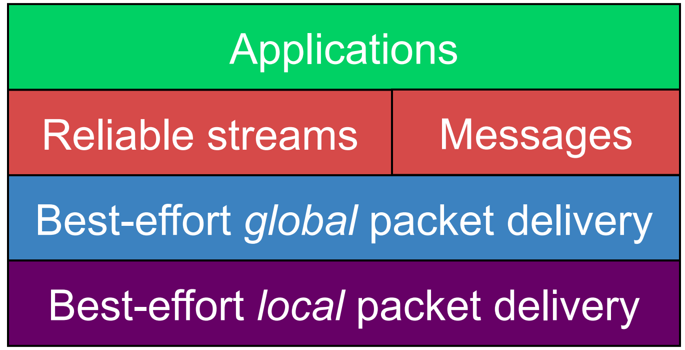


Class Meeting, Lectures 5 & 6: Transport Layer and Congestion Control

Kyle Jamieson

COS 461: Computer Networks

www.cs.princeton.edu/courses/archive/fall21/cos461

[Parts adapted from material by M. Freedman (Princeton), B. Karp (UCL), D. Katabi, (MIT), S. Shenker (UCB)]

Context: Transport Layer

- Best-effort network layer
  - drops packets
  - delays packets
  - reorders packets
  - corrupts packet contents

- Many applications want reliable transport
  - all data reach receiver, in order they were sent
  - no data corrupted
  - "reliable byte stream"

- Need a transport protocol, *e.g.*, Internet's Transmission Control Protocol (TCP)

TCP: Connection-Oriented, Reliable Byte Stream Transport

- Sending app offers stream of bytes: d0, d1, d2, …
- Receiving application sees all bytes arrive in same sequence: d0, d1, d2…
  - Not all applications need in-order behavior (*e.g.,* ssh does, but do file transfer or teleconferencing, really?)
  - result: reliable byte stream transport
- Each byte stream: *connection*, or *flow*
- Each connection uniquely identified by:
  - <sender IP, sender port, receiver IP, receiver port>

User Datagram Protocol (UDP)

- Lightweight communication between processes
  - Send and receive messages
  - Avoid overhead of ordered, reliable delivery
    - No connection setup delay, no in-kernel connection state

8 byte header

SRC port	DST port
checksum	length
DATA	

- Used by popular apps
  - Query/response for DNS
  - Some teleconferencing apps

Fundamental Problem: Ensuring At-Least-Once Delivery

- <u>A strategy</u> to ensure delivery:
  - Sender attaches a unique number (*nonce*) to each data packet sent; keeps copy of sent packet
  - Receiver returns acknowledgement (ACK) to sender for each data packet received, containing nonce
  - Sender sets a timer on each transmission
    - timer expires before ACK returns → retransmit that packet
    - ACK returns → cancel timer, discard saved copy of that packet
  - Sender limits maximum number of retransmissions

- How long should retransmit timer be?

Fundamental Problem: Estimating RTT

- Expected time of ACK's return: **round-trip time (RTT)**
  - end-to-end delay for data to reach receiver and ACK to reach sender
  - propagation delay on links
  - serialization delay at each hop
  - queuing delay at routers

- Strawman: use fixed timer (*e.g.*, 250 ms)
  - what if the route changes?
  - what if congestion occurs at one or more routers?

Estimating RTT: Exponentially Weighted Moving Average (EWMA)

- Measurements of RTT readily available
  - note time t when packet sent
  - corresponding ACK returns at time t'
  - RTT measurement = m = t'-t

EWMA weights newest samples most

How to choose α? (TCP uses 1/8)

Is mean sufficient to capture RTT behavior over time? (more later)

- Adapt over time, using EWMA:
  - measurements: $m_0, m_1, m_2, \ldots$
  - fractional weight for new measurement, α
  - $RTT_i = ((1-\alpha) \times RTT_{i-1} + \alpha \times m_i)$

Retransmission and Duplicate Delivery

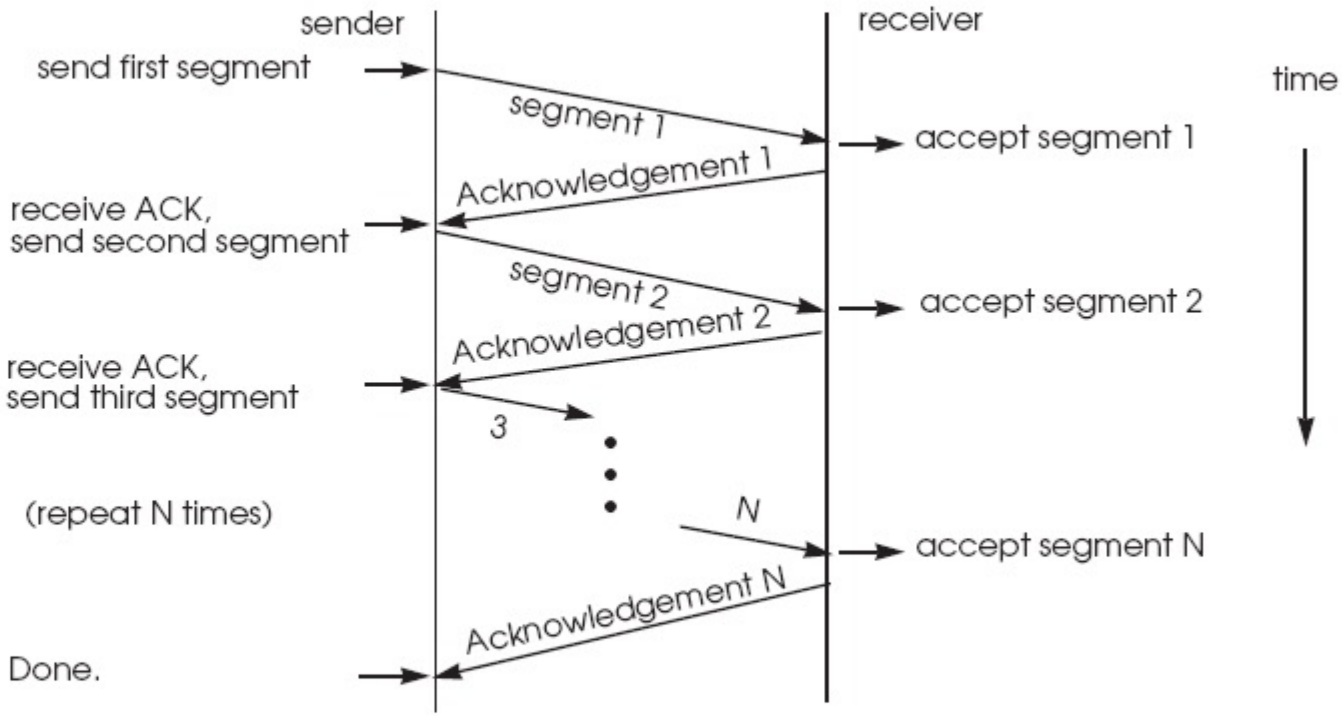
- When sender’s retransmit timer expires, two indistinguishable cases (why?):
  - data packet dropped en route to receiver, or
  - ACK dropped en route to sender
- In both cases, sender retransmits
- In latter case, duplicate data packet reaches receiver!
  - How to prevent receiver from passing duplicates to application?

Eliminating Duplicates: Exactly Once Delivery

- Each packet sent with unique identifier (*nonce*)
- Strawman: receiver stores nonces previously seen (*tombstones*)
  - if received packet seen before, drop, but resend ACK to sender
- How many tombstones must receiver store?

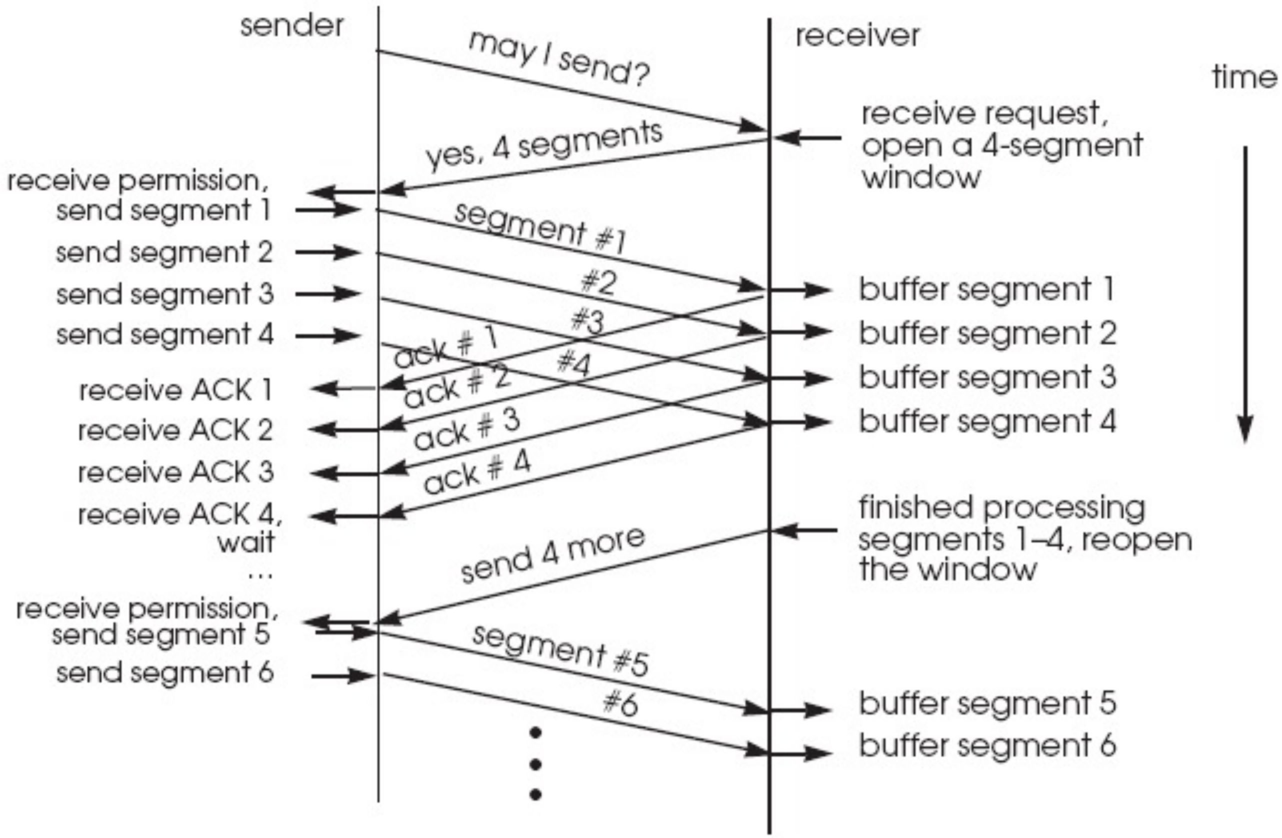
- Better plan: **sequence numbers**
  - sender marks each packet with **monotonically increasing** *sequence number* (non-random nonce)
  - sender includes greatest ACKed sequence number in its packets
  - receiver remembers only greatest received sequence number, drops received packets with smaller ones

Window-Based Flow Control: Motivation



- Suppose sender sends one packet, awaits ACK, repeats...
- Result: one packet sent per RTT
- *e.g.*, 70 ms RTT, 1500-byte packets: **Max throughput: 171 Kbps**

Fixed Window-Based Flow Control

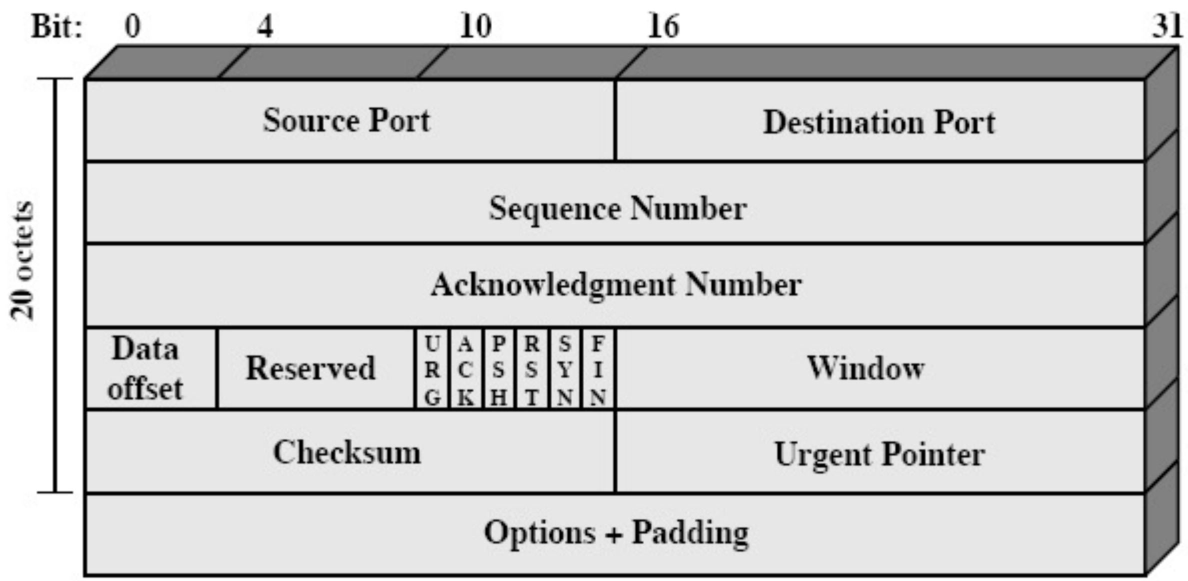


- Pipeline transmissions to "keep pipe full"; overlap ACKs with data
- Sender sends **window** of packets sequentially, without awaiting ACKs
- Sender retains packets until ACKed, tracks which have been ACKed
- Sender sets retransmit timer for each window; when expires, resends all unACKed packets in window

Choosing Window Size: Bandwidth-Delay Product

- How large a window is required at sender to keep the pipe full?
- Network bottleneck: point of slowest rate along path between sender and receiver
- To keep pipe full
  - window size ≥ RTT × bottleneck rate
- Window too small: can't fill pipe
- Window too large: unnecessary network load/queuing/loss

TCP Packet Header



- TCP packet: IP header + TCP header + data
- TCP header: 20 bytes long
- Checksum covers header + "pseudo header"
  - IP header source and destination addresses, protocol
  - Length of TCP segment (TCP header + data)

TCP Header Details

- Connections inherently bidirectional; all TCP headers carry both data and ACK sequence numbers
- 32-bit sequence numbers are in units of bytes
- Source and destination ports
  - multiplexing of TCP by applications
  - UNIX: local ports below 1024 reserved (only root may use them)
- Window: advertisement of number of bytes advertiser willing to accept

TCP Connection Establishment: Motivation

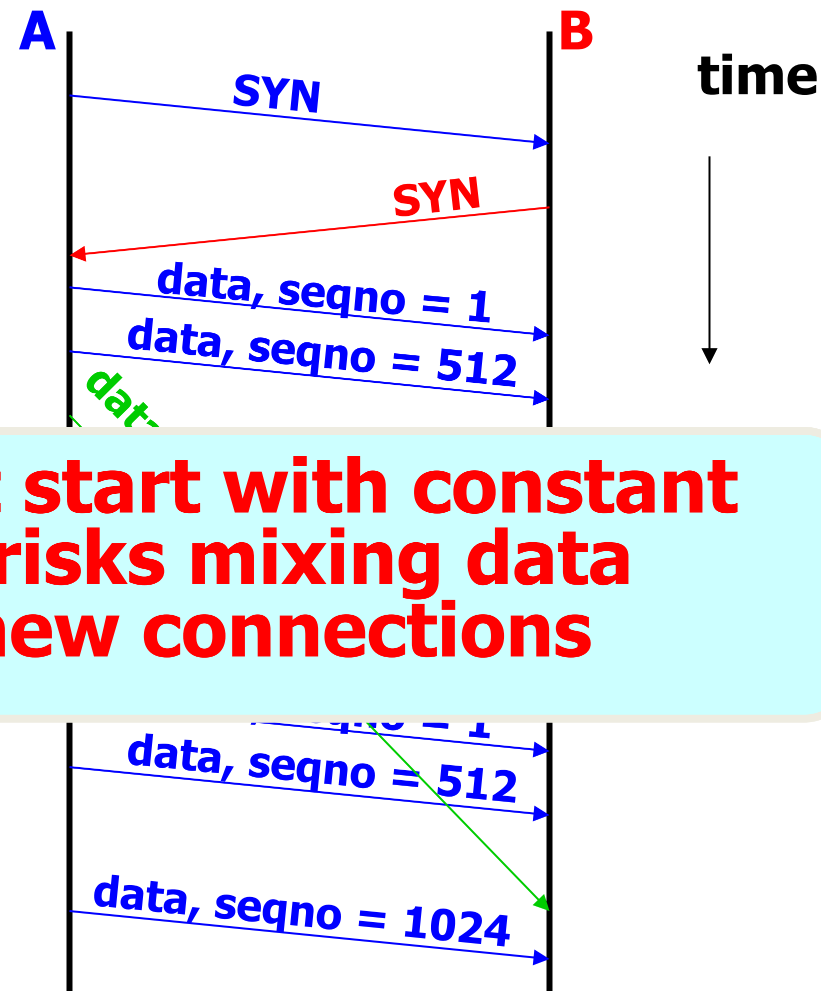
- Goals:
  - Start TCP connection between two hosts
  - Avoid mixing data from old connection in new connection
  - Avoid confusing previous connection attempts with current one
  - Prevent (most) third parties from impersonating **(spoofing)** one endpoint
- SYN packets (SYN flag in TCP header set) used to establish connections
- Use retransmission timer to recover from lost SYNs
- What protocol meets above goals?

TCP Connection Establishment: Non-Solution (I)

- Use two-way handshake
- A sends SYN to B
  - A retransmits SYN if not received
  - B accepts by returning SYN to A

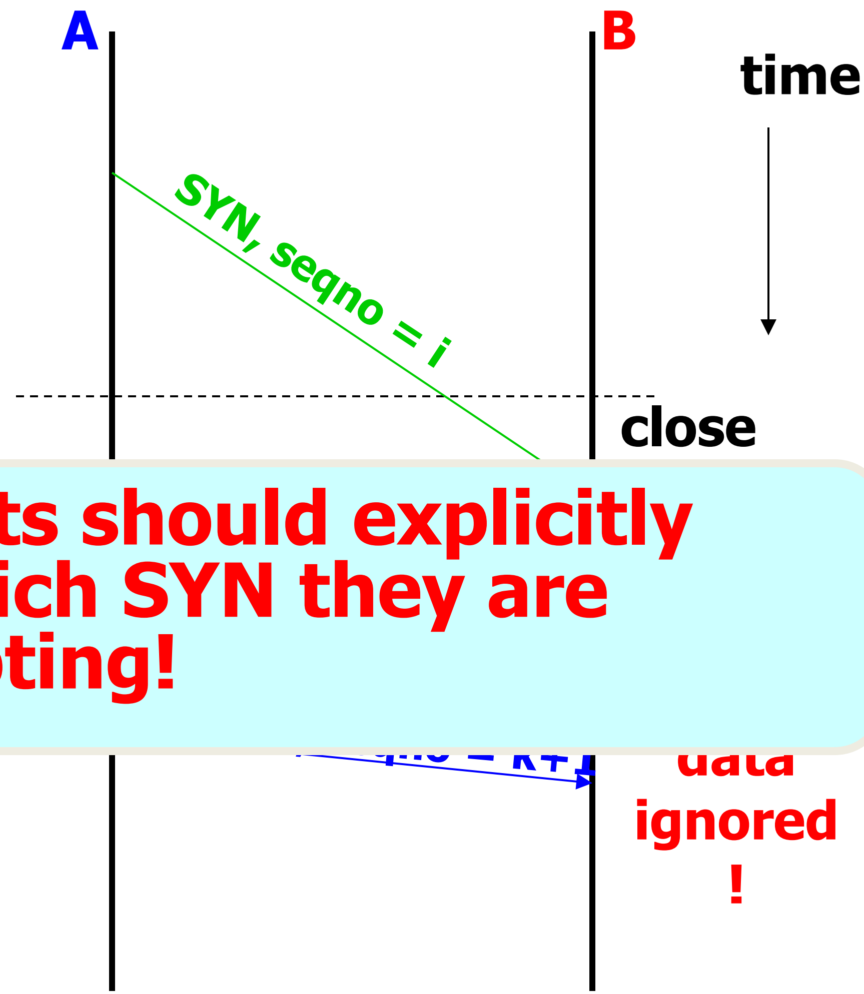
Connections shouldn't start with constant sequence number; risks mixing data between old and new connections

- What about delayed data packets from old connection?



TCP Connection Establishment: Non-Solution (II)

- Two-way handshake, as before
- But enclose random initial sequence numbers on SYNs



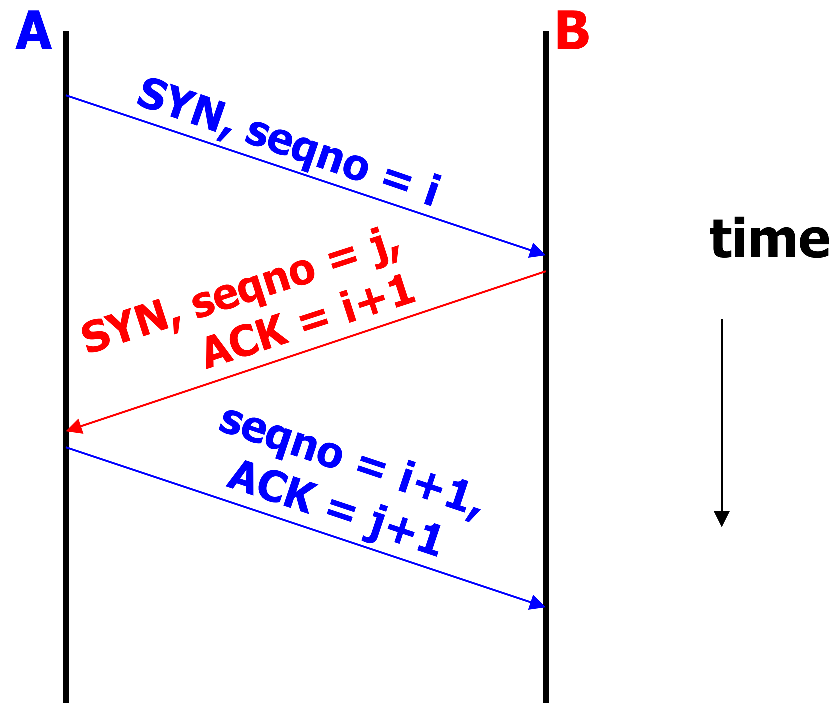
Connection attempts should explicitly acknowledge which SYN they are accepting!

connection successfully established

– **B will drop all of A's data!**

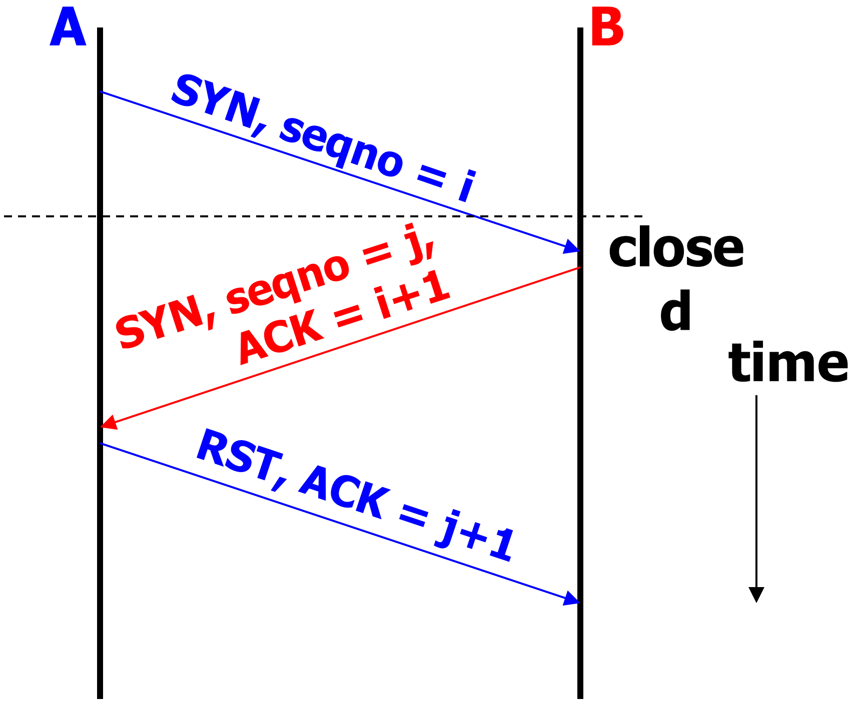
TCP Connection Establishment: 3-Way Handshake

- Set SYN on connection request
- Each side chooses random *initial sequence number* (ISN)
- Each side explicitly ACKs the sequence number of the SYN it's responding to



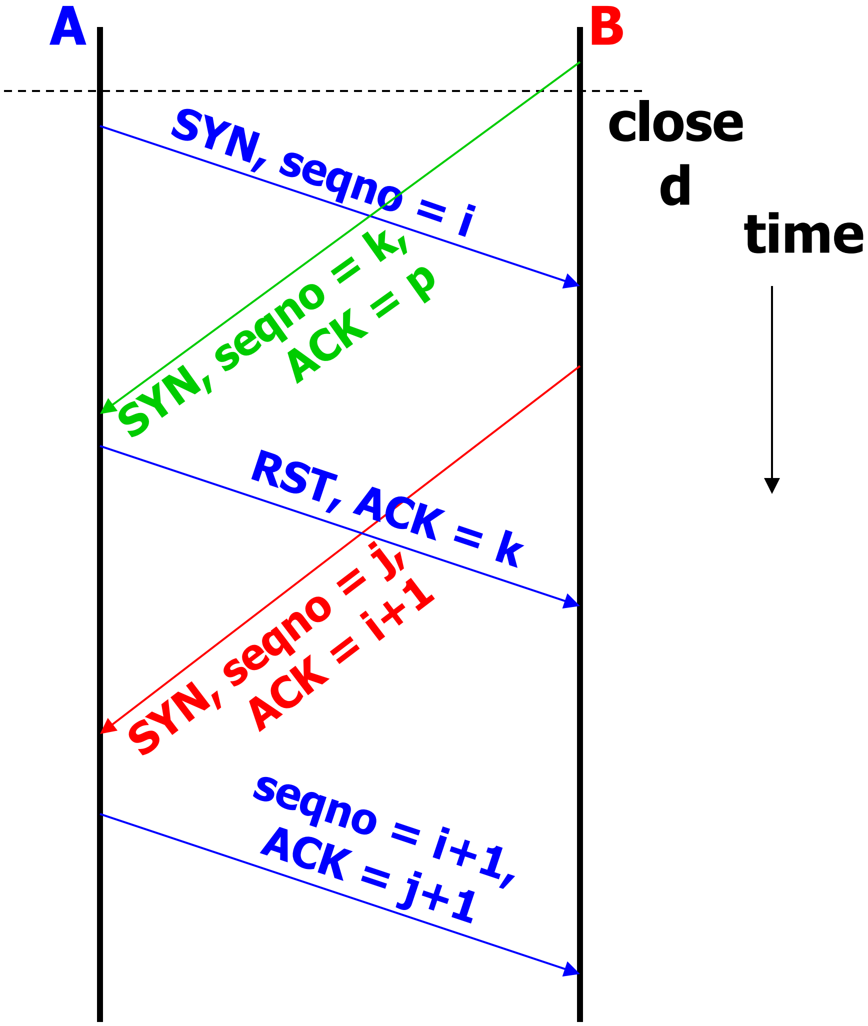
Robustness of 3-Way Handshake: Delayed SYN

- Suppose A's SYN i delayed, arrives at B after connection closed
- B responds with SYN/ACK for i+1
- A doesn't recognize i+1; responds with **reset,** RST flag set in TCP header
- A rejects connection



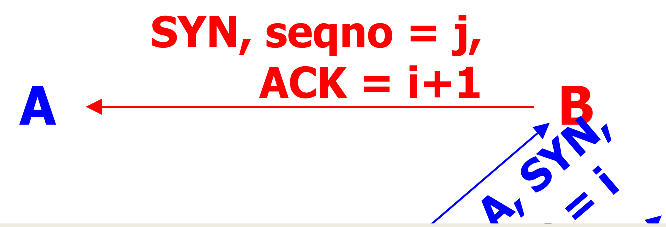
Robustness of 3-Way Handshake: Delayed SYN/ACK

- A attempts connection to B
- Suppose B's SYN k/ACK p delayed, arrives at A during new connection attempt
- A rejects SYN k; sends RST to B
- Connection from A to B succeeds unimpeded



Robustness of 3-Way Handshake: Source Spoofing

- Suppose host B trusts host A, based on A's IP
  - e.g., B allows any account creation request from A



Unless he is on path between A and B, adversary cannot spoof A to B or vice-versa!

Why: random ISNs on SYNs

"create an account l33thax0r")

- Can M establish a connection to B as A?

Next Up in 461

Next Class Meeting

Lectures 7 (Queue Management) and

8 (Middleboxes, Tunneling)

Precepts this Thursday and Friday