NANJING UNIVERSITY

运输层

Outline

- UDP: User Datagram Protocol
- TCP: Transmission Control Protocol
- TCP Connection Setup
- TCP Connection Teardown

UDP: User Datagram Protocol

- Lightweight communication between processes
  - Avoid overhead and delays of order & reliability
- UDP described in RFC 768 – (1980!)
  - Destination IP address and port to support demultiplexing

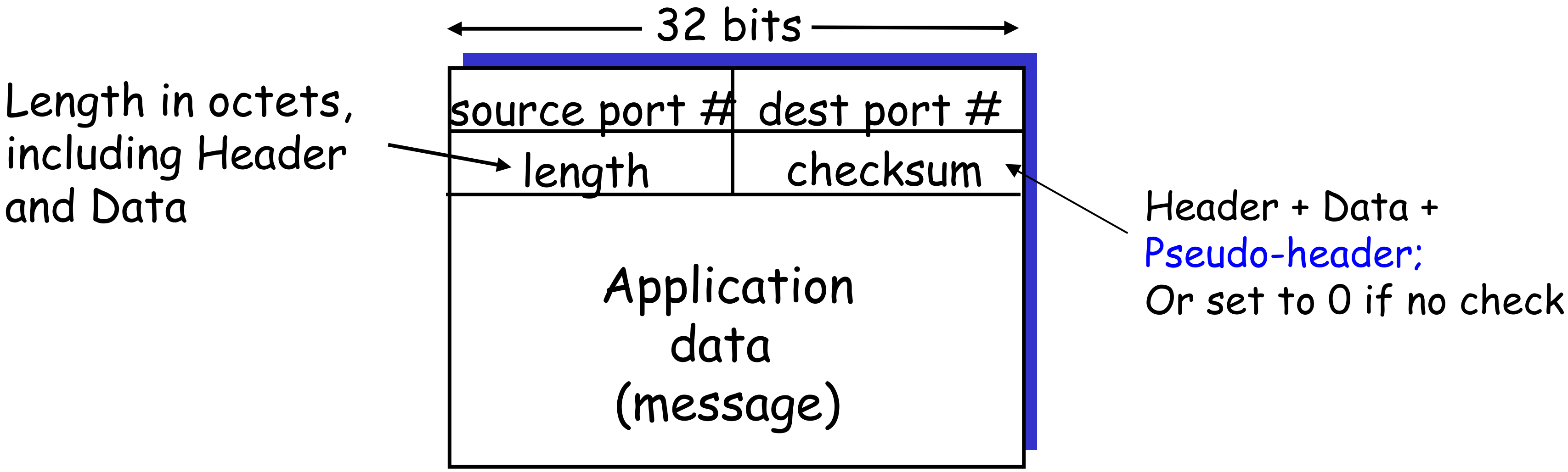
UDP (cont'd)

- "Best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- Connectionless:
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others
- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP

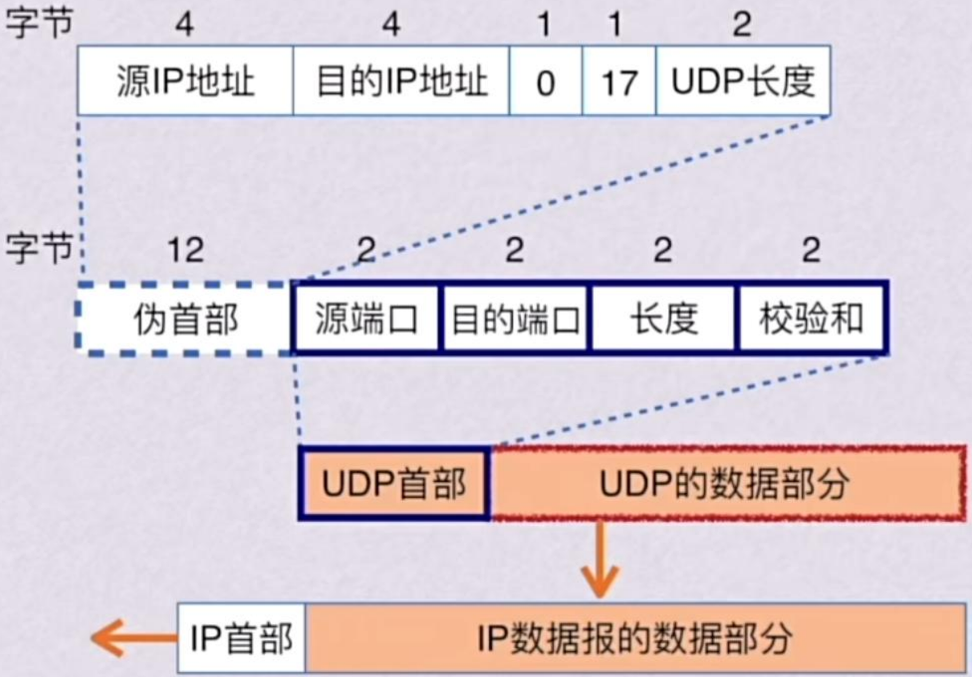
Why is there a UDP?

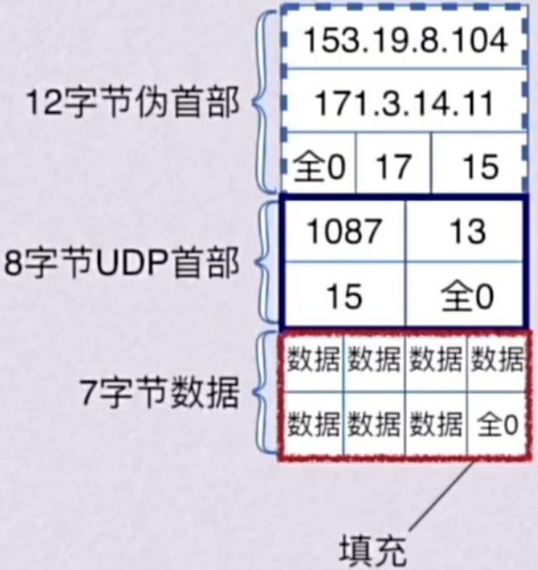
- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control: UDP can blast away as fast as desired

UDP Segment Format



UDP的首部格式





伪首部是指这个首部不属于UDP数据报真正的首部，而只是在计算校验和时临时添加在UDP用户数据报前面伪首部既不向下传送也不向上递交，仅仅是为了计算校验和

UDP把首部和数据部分一起检验，和IP不同（IP的校验和只检测首部）

UDP checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

sender:

- treat segment contents, including header fields, as sequence of 16-bit integers
- checksum: addition of segment contents, and its complement sum
- sender puts checksum value into UDP checksum field

receiver:

- compute checksum of received segment
- check if the sum of computed checksum and checksum field value equals 1111….1111:
  - NO - error detected
  - YES - no error detected. But maybe errors nonetheless?

Internet checksum: example

example: add two 16-bit integers

		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

Optional error checking

- Optional error checking on the packet contents
  - (checksum field = 0 means "don't verify checksum")
  - See text on how checksums are calculated
- Source port is also optional
  - Useful to respond back to the sender in some cases

Outline

- UDP: User Datagram Protocol
- TCP: Transmission Control Protocol
- TCP Connection Setup
- TCP Connection Teardown

The TCP Abstraction

- TCP delivers a reliable, in-order, byte stream
- Reliable: TCP resends lost packets (recursively)
  - Until it gives up and shuts down connection
- In-order: TCP only hands consecutive chunks of data to application
- Byte stream: TCP assumes there is an incoming stream of data, and attempts to deliver it to app

What does TCP use from what we've seen so far?

- Most of what we've seen
  - Checksums
  - Sequence numbers are byte offsets
  - Sender and receiver maintain a sliding window
  - Receiver sends cumulative acknowledgements (like GBN)
    - ✓ Sender maintains a single retransmission timer
  - Receivers buffer out-of-sequence packets (like SR)
- Few more: fast retransmit, timeout estimation algorithms etc.

TCP header

Used to Mux and Demux

<table>
<tr><td colspan="2">Source port</td><td colspan="2">Destination port</td></tr>
<tr><td colspan="4">Sequence number</td></tr>
<tr><td colspan="4">Acknowledgment</td></tr>
<tr><td>HdrLen</td><td>0</td><td>Flags</td><td>Advertised window</td></tr>
<tr><td colspan="3">Checksum</td><td>Urgent pointer</td></tr>
<tr><td colspan="4">Options (variable)</td></tr>
<tr><td colspan="4">Data</td></tr>
</table>

TCP header

Computed
over pseudo-header
and data

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

What does TCP do?

- Most of what we've seen
  - Checksum
  - Sequence numbers are byte offsets

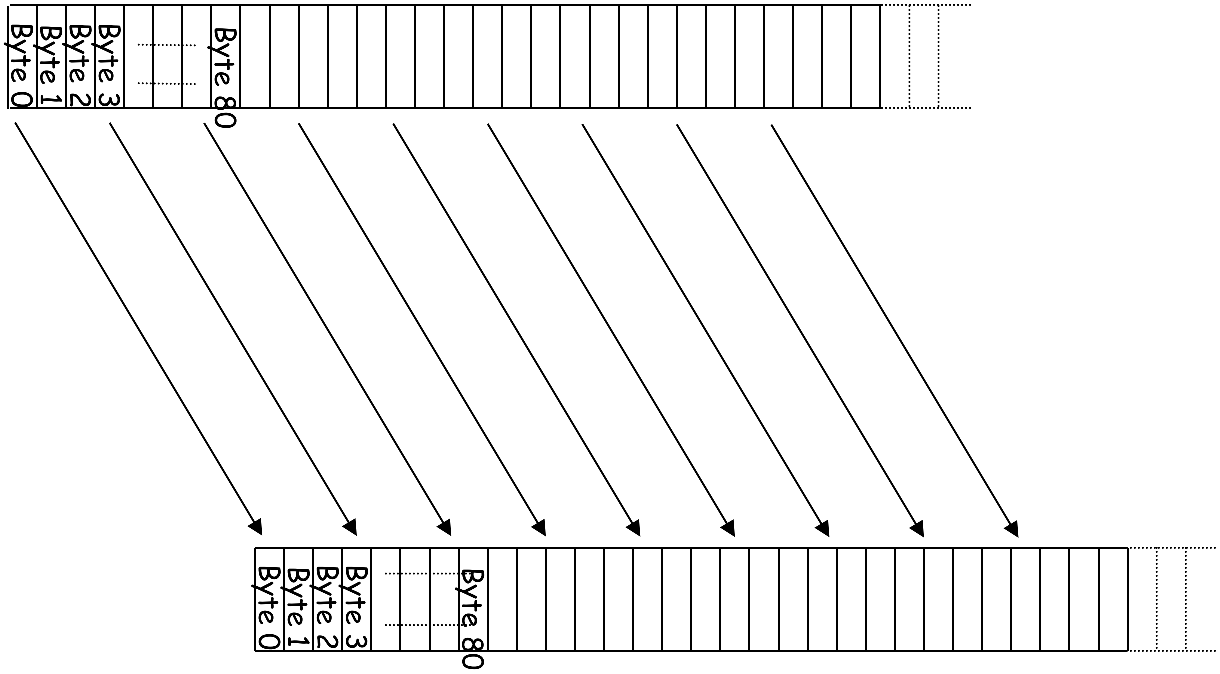
TCP header

Byte offsets (NOT packet id), because TCP is a byte stream

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

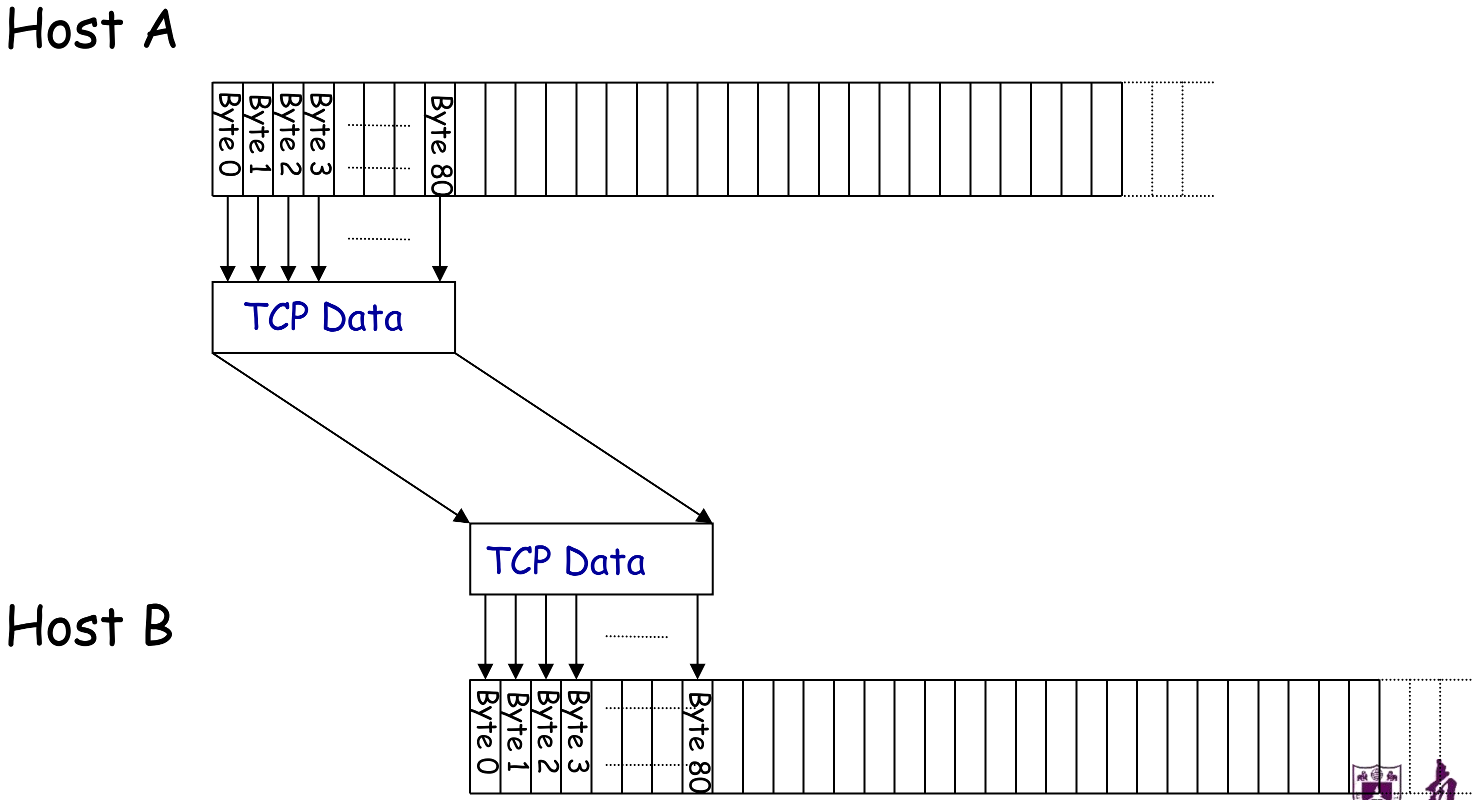
TCP "stream of bytes" service...

Application @ Host A

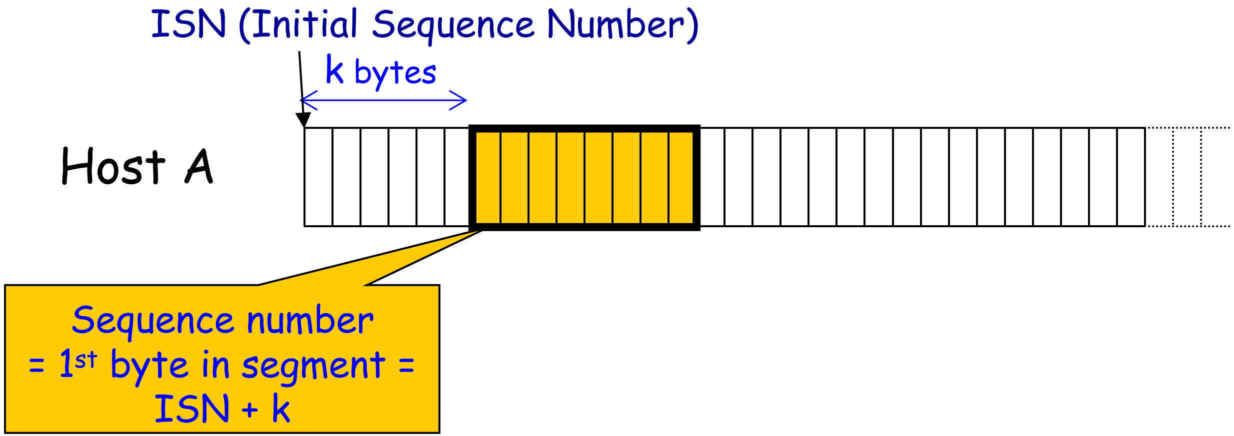


Application @ Host B

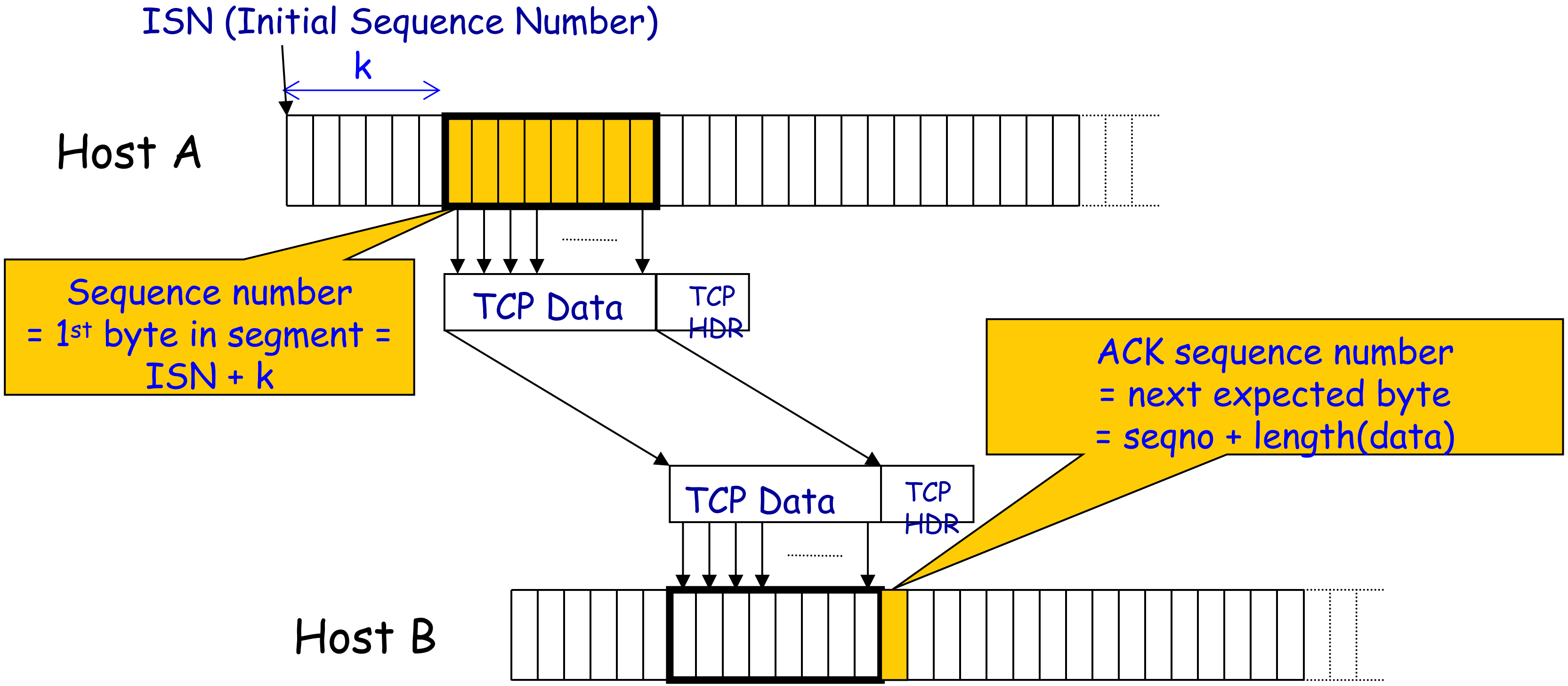
TCP "stream of bytes" service...



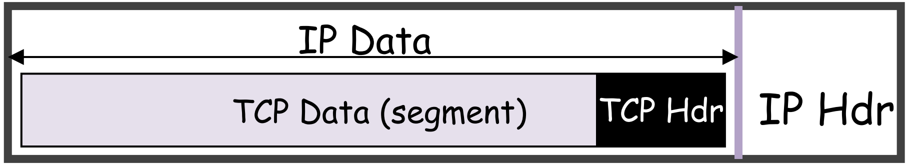
Sequence numbers



Sequence numbers



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)

TCP header

Starting byte offset of data carried in this segment

<table>
<tr><td colspan="3">Source port</td><td>Destination port</td></tr>
<tr><td colspan="4">Sequence number</td></tr>
<tr><td colspan="4">Acknowledgment</td></tr>
<tr><td>HdrLen</td><td>0</td><td>Flags</td><td>Advertised window</td></tr>
<tr><td colspan="3">Checksum</td><td>Urgent pointer</td></tr>
<tr><td colspan="4">Options (variable)</td></tr>
<tr><td colspan="4">Data</td></tr>
</table>

What does TCP do?

- Most of what we've seen
  - Checksum
  - Sequence numbers are byte offsets
  - Receiver sends cumulative acknowledgements (like GBN)

ACKs 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 in-order byte received is Y s.t. (Y+1) < X
    - ✓ ACK acknowledges Y+1
    - ✓ Even if this has been ACKed before

Typical operation

- Sender: seqno=X, length=B
- Receiver: ACK=X+B
- Sender: seqno=X+B, length=B
- Receiver: ACK=X+2B
- Sender: seqno=X+2B, length=B

- Seqno of next packet is same as last ACK field

Loss with cumulative ACKs

- Sender sends packets with 100B and seqnos.:
  - 100, 200, 300, 400, 500, 600, 700, 800, 900, …
- Assume the fifth packet (seqno 500) is lost, but no others
- Stream of ACKs will be:
  - 200, 300, 400, 500 (seqno:600), 500 (seqno:700), 500 (seqno:800), 500 (seqno:900),…

TCP header

Acknowledgment gives seqno just beyond highest seqno received **in order**

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

What does TCP do?

- Most of what we've seen
  - Checksum
  - Sequence numbers are byte offsets
  - Receiver sends cumulative acknowledgements (like GBN)
  - Receivers can buffer out-of-sequence packets (like SR)

What does TCP introduce?

- Most of what we've seen
  - Checksum
  - Sequence numbers are byte offsets
  - Receiver sends cumulative acknowledgements (like GBN)
  - Receivers can buffer out-of-sequence packets (like SR)

- Introduces fast retransmit: duplicate ACKs trigger early retransmission

Loss with cumulative ACKs

- Duplicate ACKs are a sign of an isolated loss
  - The lack of ACK progress means 500 hasn't been delivered
  - Stream of ACKs means some packets are being delivered
- Trigger retransmission upon receiving k duplicate ACKs
  - TCP uses k=3
  - Faster than waiting for timeout

Loss with cumulative ACKs

- Two choices after resending:
  - ➢ Send missing packet and move sliding window by the number of dup ACKs
    - ✓ Speeds up transmission, but might be wrong
  - ➢ Send missing packet, and wait for ACK to move sliding window
    - ✓ Is slowed down by single dropped packets

- Which should TCP do?

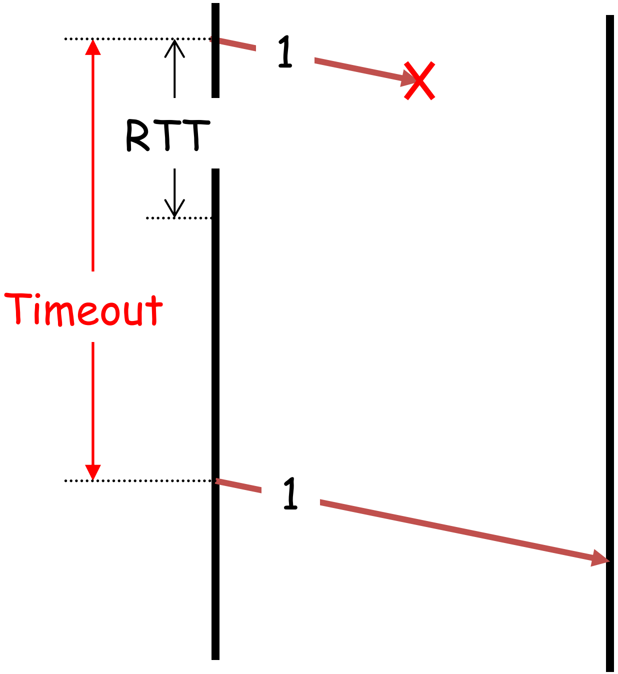
What does TCP introduce?

- Most of what we've seen
  - Checksum
  - Sequence numbers are byte offsets
  - Receiver sends cumulative acknowledgements (like GBN)
  - Receivers buffer out-of-sequence packets (like SR)
- Introduces fast retransmit: duplicate ACKs trigger early retransmission
- Sender maintains a single retransmission timer (like GBN) and retransmits on timeout

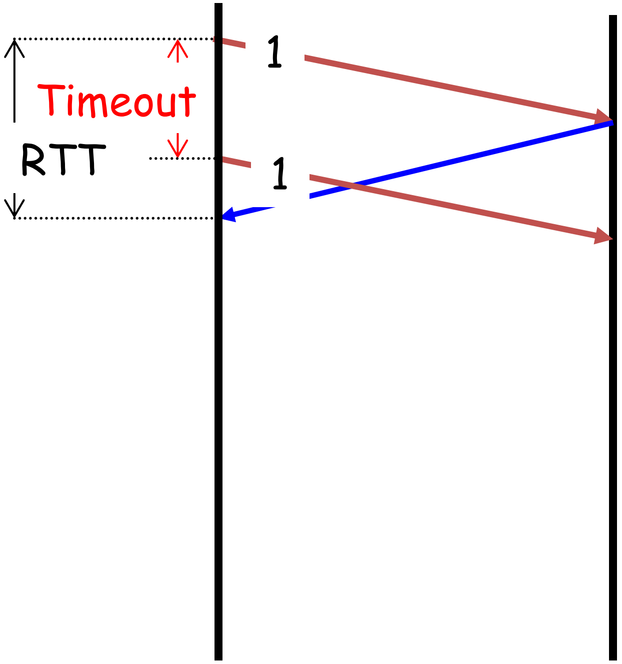
Retransmission timeout

- If the sender hasn't received an ACK by timeout, retransmit the first packet in the window
- How do we pick a timeout value?

Timing illustration



Timeout too long → inefficient



Timeout too short → duplicate packets

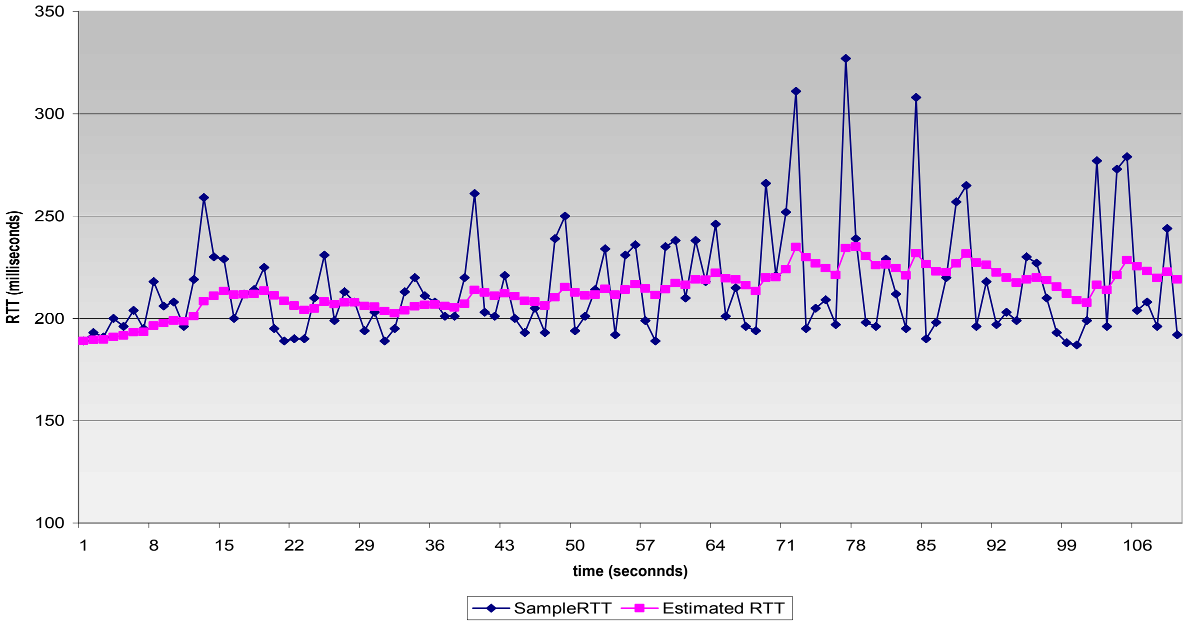
Retransmission timeout

- If the sender hasn't received an ACK by timeout, retransmit the first packet in the window
- How to set timeout?
  - Too long: connection has low throughput
  - Too short: retransmit packet that was just delayed
- Solution: make timeout proportional to RTT
  - But how do we measure RTT?

RTT estimation

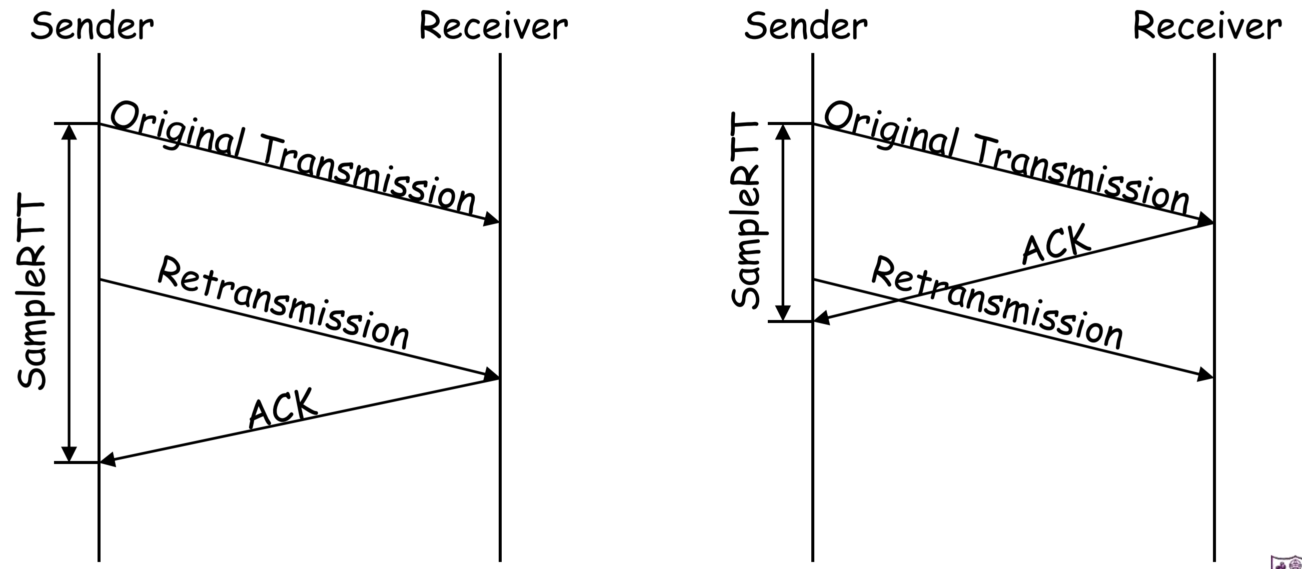
- Exponential weighted average of RTT samples

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



Problem: Ambiguous measurements

- How do we differentiate between the real ACK, and ACK of the retransmitted packet?



Karn/Partridge algorithm

- Don't use SampleRTT from retransmissions
  - Once retransmitted, ignore that segment in the future
- Computes EstimatedRTT using $\alpha = 0.125$
- Timeout value (RTO) $= 2 \times$ EstimatedRTT
  - Employs exponential backoff
    - ✓ Every time RTO timer expires, set RTO $\leftarrow 2 \cdot$RTO
      - (Up to maximum $\geq$ 60 sec)
    - ✓ Every time new measurement comes in (= successful original transmission), collapse RTO back to $2 \times$ EstimatedRTT
- Insensitive to RTT variations

NANJING UNIVERSITY

Jacobson/Karels algorithm

- Problem: need to better capture variability in RTT
  - Directly measure deviation
- Deviation = | SampleRTT – EstimatedRTT |
- DevRTT: exponential average of Deviation
- RTO = EstimatedRTT + 4 x DevRTT

$$SRTT(k+1) = (1-g) \times SRTT(k) + g \times RTT(k+1)$$

$$SERR(k+1) = RTT(k+1) - SRTT(k)$$

$$SDEV(k+1) = (1-h) \times SDEV(k) + h \times \left|SERR(k+1)\right|$$

$$RTO(k+1) = SRTT(k+1) + f \times SDEV(k+1)$$

$$g = 1/8 = 0.125 \quad h = 1/4 = 0.25 \quad f = 2 \text{ or } 4$$

TCP header

Number of 4-byte words in the header

<table>
<tr><td colspan="3">Source port</td><td>Destination port</td></tr>
<tr><td colspan="4">Sequence number</td></tr>
<tr><td colspan="4">Acknowledgment</td></tr>
<tr><td>HdrLen</td><td>0</td><td>Flags</td><td>Advertised window</td></tr>
<tr><td colspan="3">Checksum</td><td>Urgent pointer</td></tr>
<tr><td colspan="4">Options (variable)</td></tr>
<tr><td colspan="4">Data</td></tr>
</table>

Outline

- UDP: User Datagram Protocol
- TCP: Transmission Control Protocol
- TCP Connection Setup
- TCP Connection Teardown

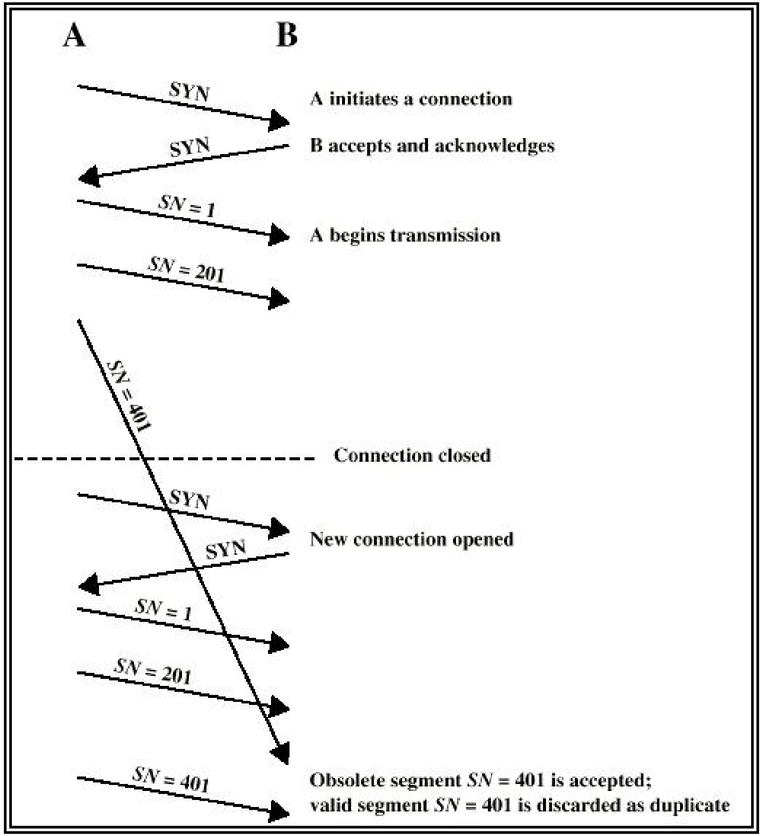
- TCP header field for connection establishment and teardown

<table>
<tr><td colspan="8">Source Port</td><td colspan="2">Destination Port</td></tr>
<tr><td colspan="10">Sequence Number</td></tr>
<tr><td colspan="10">Acknowledgement Number</td></tr>
<tr><td>Data Offset</td><td>Reserved</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="2">Window</td></tr>
<tr><td colspan="8">Checksum</td><td colspan="2">Urgent Pointer</td></tr>
<tr><td colspan="9">TCP Options</td><td>Padding</td></tr>
<tr><td colspan="10">Data</td></tr>
</table>

Connection Establishment

- 2-way handshake
  - A sends SYN, B replies with SYN
  - Lost SYNs handled by re-transmission
  - Ignore duplicate SYNs once connected
- Problem
  - How to recognize slipped segments from old connection
  - How to recognize duplicated obsolete SYN

2-Way Handshake: Slipped Data Segment

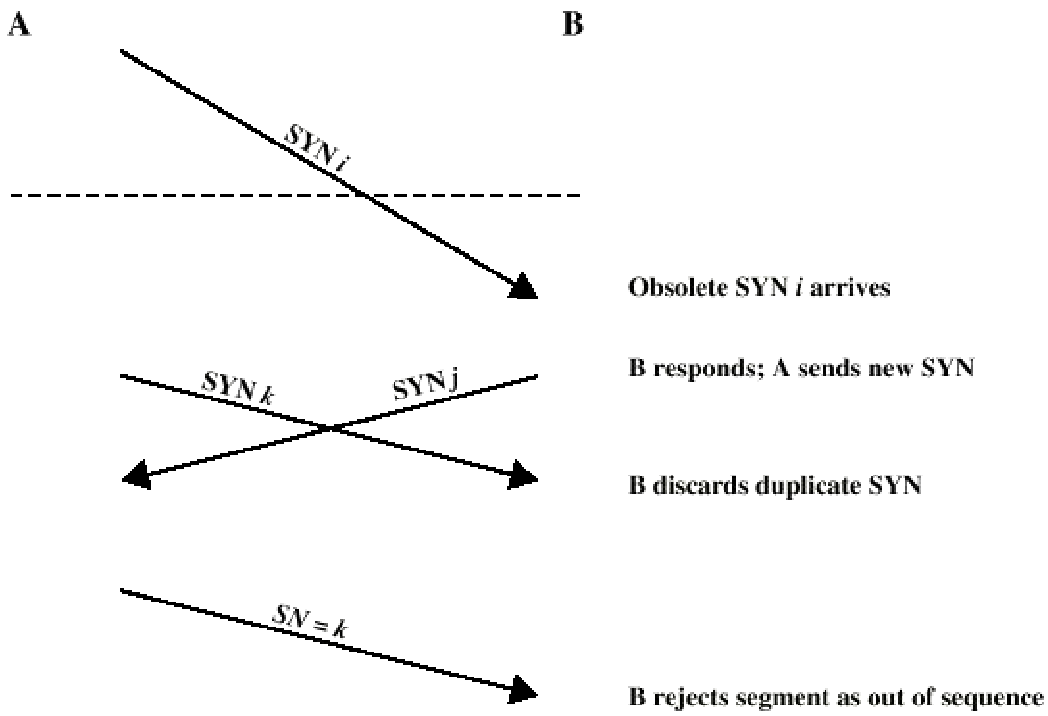


Initial Sequence Number (ISN)

- Handle
  - Start each new connection with a different initial sequence number (ISN) far from previous connection
  - The connection request is of the form SYN i+1, where i is the sequence number of the first data segment that will be sent on this connection.

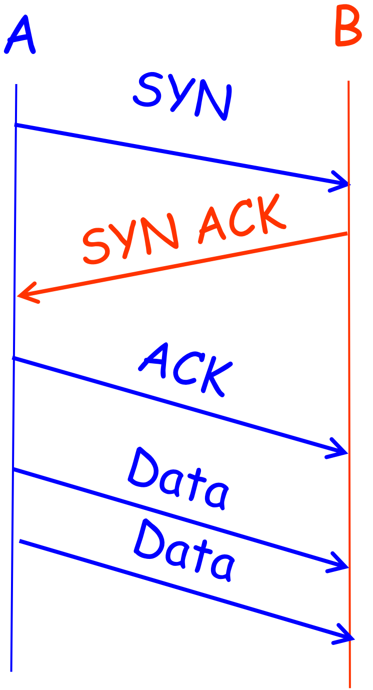
- However:

2-Way Handshake: Obsolete SYN



Solution: three-way handshake

- Three-way handshake to establish connection
  - Host A sends a SYN (open; "synchronize sequence numbers") to host B
  - Host B returns a SYN acknowledgment (SYN ACK)
  - Host A sends an ACK to acknowledge the SYN ACK



三方握手：确认对方的SYN和序号

TCP header

Flags:
<u>SYN</u>
<u>ACK</u>
FIN
RST
PSH
URG

<table>
<tr><td colspan="3">Source port</td><td colspan="2">Destination port</td></tr>
<tr><td colspan="5">Sequence number</td></tr>
<tr><td colspan="5">Acknowledgment</td></tr>
<tr><td>HdrLen</td><td>0</td><td>Flags</td><td colspan="2">Advertised window</td></tr>
<tr><td colspan="3">Checksum</td><td colspan="2">Urgent pointer</td></tr>
<tr><td colspan="5">Options (variable)</td></tr>
<tr><td colspan="5">Data</td></tr>
</table>

NANJING UNIVERSITY

Step 1: A's initial SYN packet

A tells B to open a connection

A's port			B's port
A's Initial Sequence Number			
N/A			
5	0	SYN	Advertised window
Checksum			Urgent pointer

Step 1: B's SYN-ACK packet

B tells it accepts and is ready to accept next packet

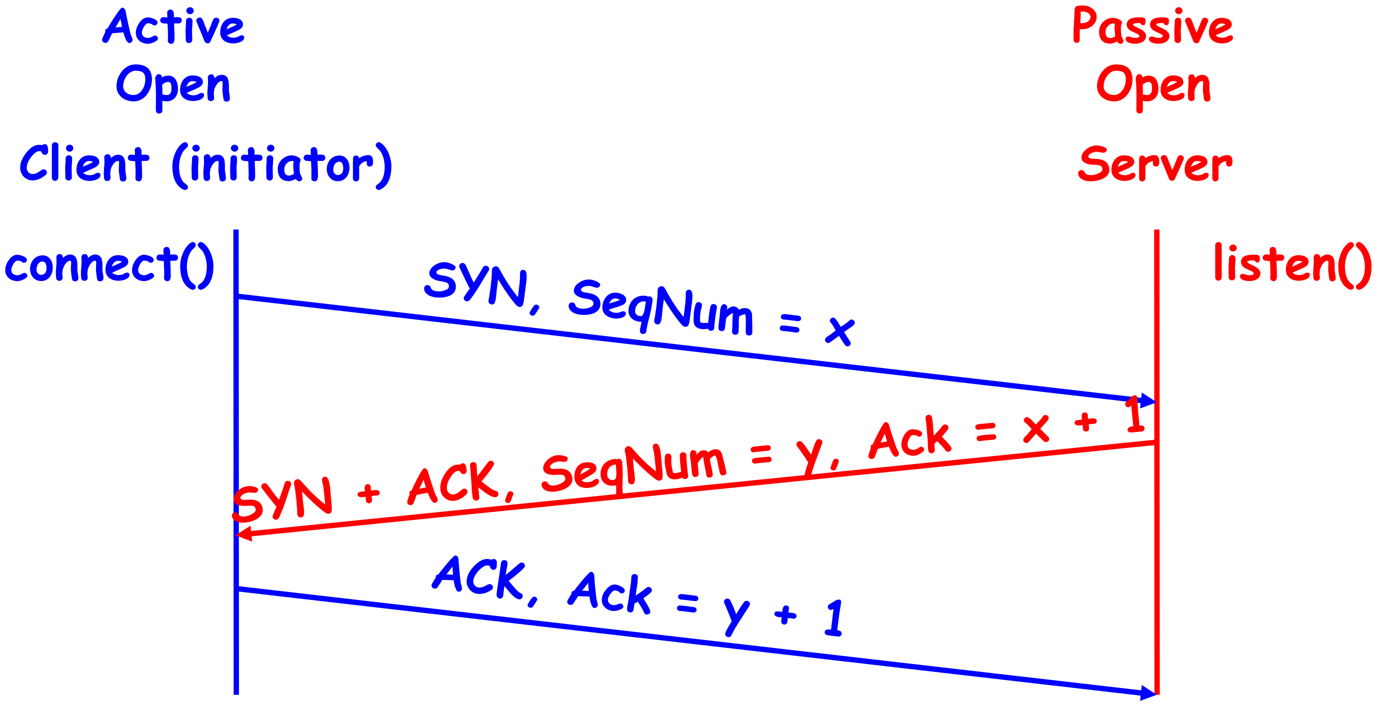
<table>
<tr><td colspan="3">B's port</td><td>A's port</td></tr>
<tr><td colspan="4">B's Initial Sequence Number</td></tr>
<tr><td colspan="4">ACK=A's ISN+1</td></tr>
<tr><td>5</td><td>0</td><td>SYN|ACK</td><td>Advertised window</td></tr>
<tr><td colspan="3">Checksum</td><td>Urgent pointer</td></tr>
</table>

Step 1: A's ACK to SYN-ACK

A tells B to open a connection

<table>
<tr><td colspan="3">A's port</td><td>B's port</td></tr>
<tr><td colspan="4">A's Initial Sequence Number</td></tr>
<tr><td colspan="4">ACK=B's ISN+1</td></tr>
<tr><td>5</td><td>0</td><td>ACK</td><td>Advertised window</td></tr>
<tr><td colspan="3">Checksum</td><td>Urgent pointer</td></tr>
</table>

TCP's 3-Way handshaking



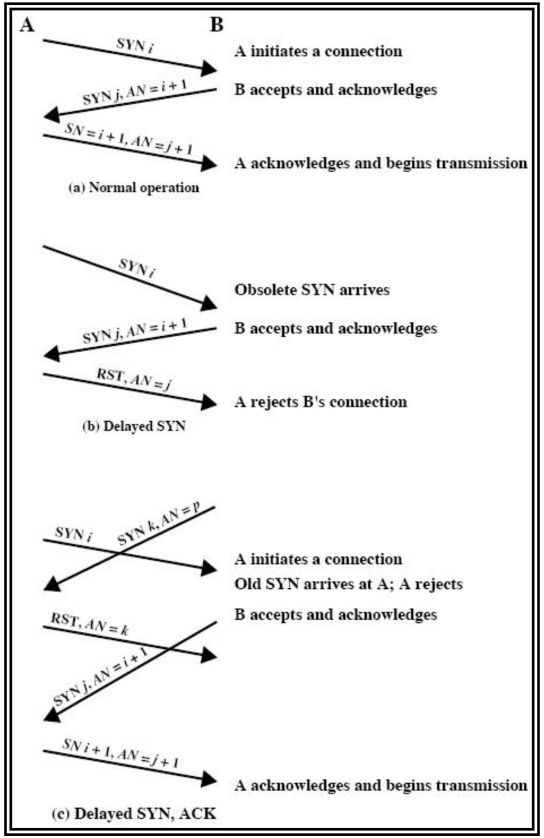
What if the SYN Packet Gets Lost?

- Suppose the SYN packet gets lost
  - Packet dropped by the network or server is busy
- Eventually, no SYN-ACK arrives
  - Sender retransmits the SYN on timeout
- 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
    - ✓ Some implementations instead use 6 seconds

SYN loss and web downloads

- User clicks on a hypertext link
  - ➢ Browser creates a socket and does a "connect"
  - ➢ The "connect" triggers the OS to transmit a SYN
- If the SYN is lost…
  - ➢ 3-6 seconds of delay: can be very long
  - ➢ User may become impatient and can retry
- User triggers an "abort" of the "connect"
  - ➢ Browser creates a new socket and another "connect"
  - ➢ Can be effective in some cases

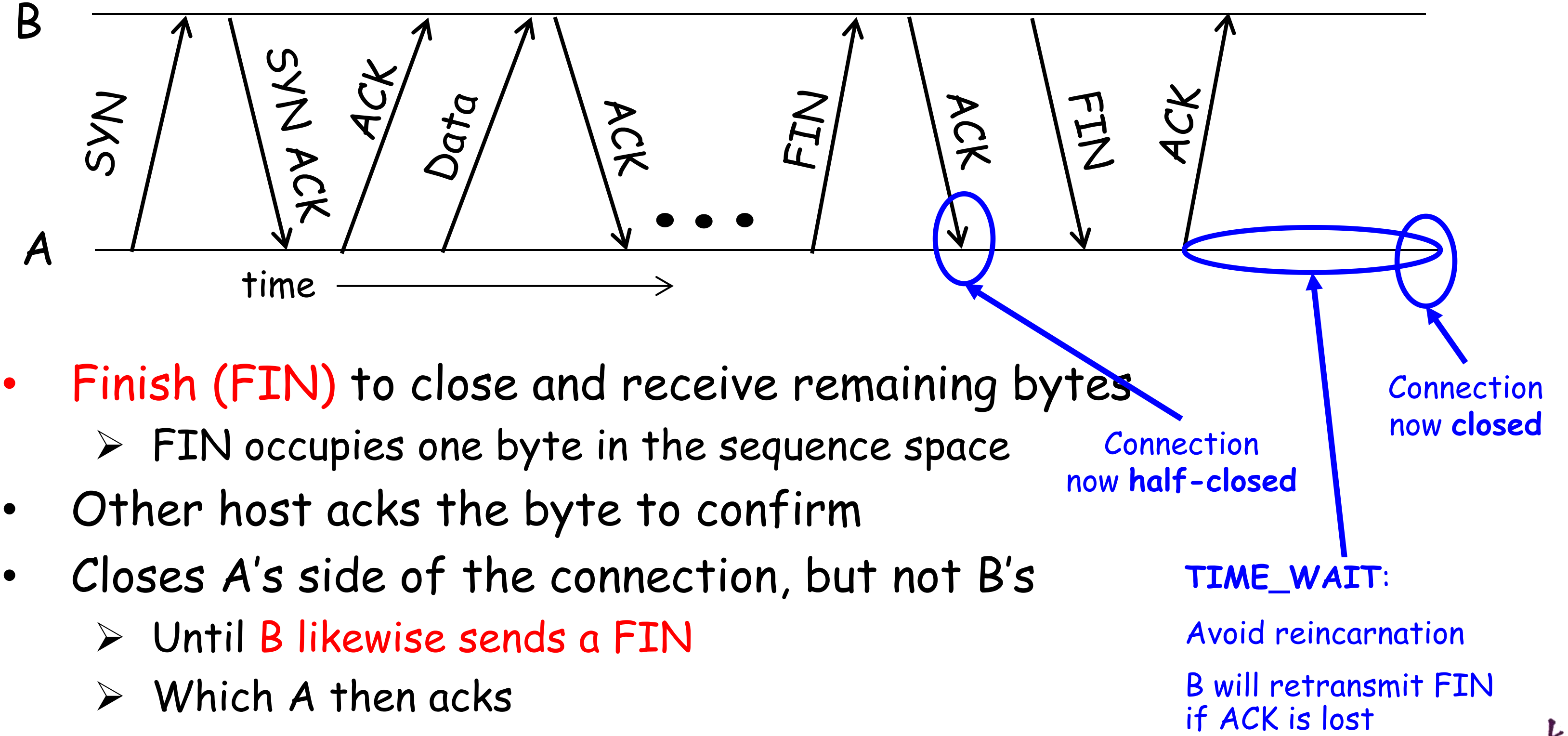
Three-Way Handshake: Examples



Outline

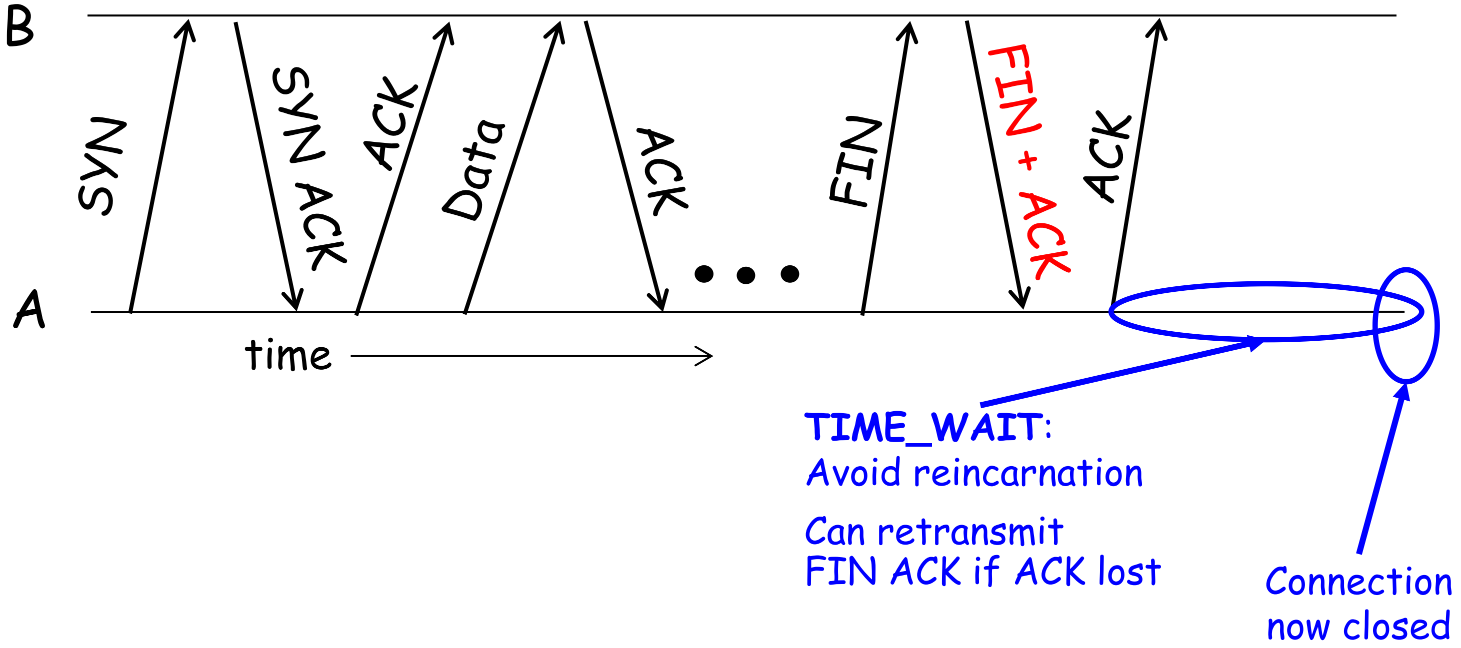
- UDP: User Datagram Protocol
- TCP: Transmission Control Protocol
- TCP Connection Setup
- TCP Connection Teardown

Normal termination, one side at a time



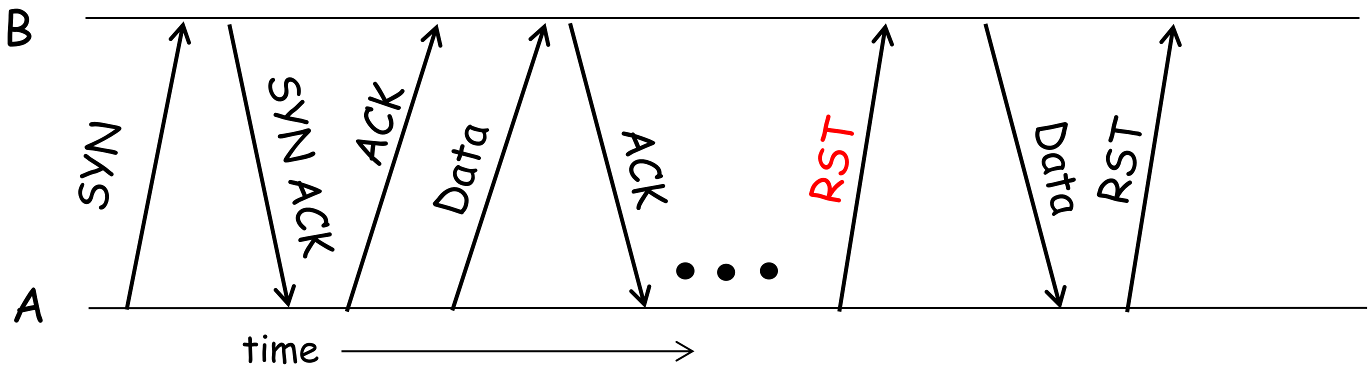
- Finish (FIN) to close and receive remaining bytes
  - FIN occupies one byte in the sequence space
- Other host acks the byte to confirm
- Closes A's side of the connection, but not B's
  - Until B likewise sends a FIN
  - Which A then acks

Normal termination, both together



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

Abrupt termination



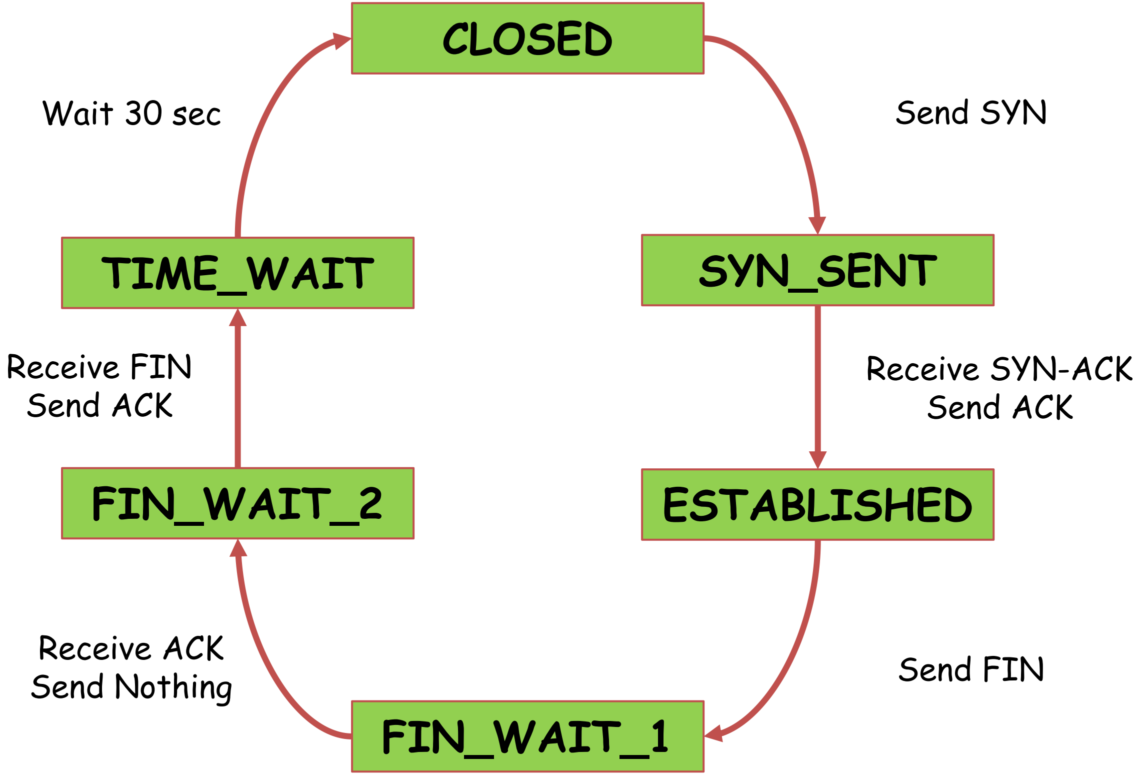
- A sends a RESET (RST) to B
  - E.g., because application process on A crashed
- That's it
  - B does not ack the RST
  - Thus, RST is not delivered reliably, and any data in flight is lost
  - But: if B sends anything more, will elicit another RST

TCP header

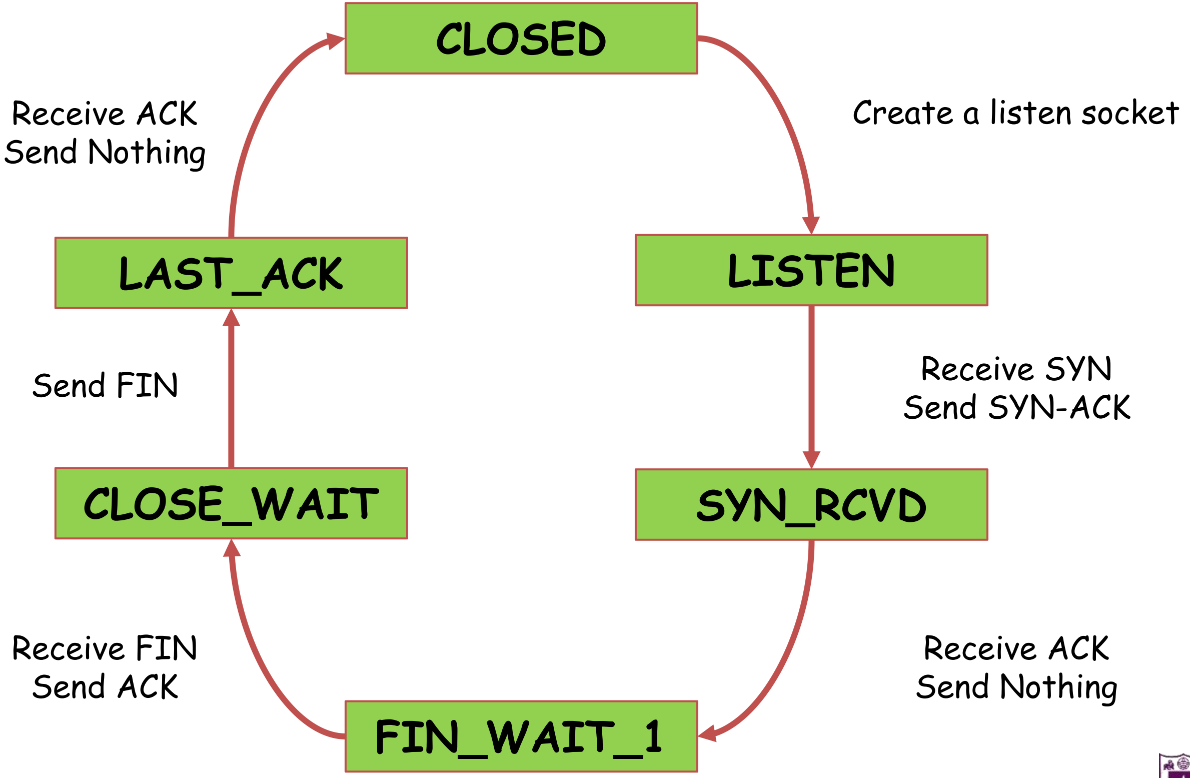
Flags:
SYN
ACK
FIN
RST
PSH
URG

<table>
<tr><td colspan="3">Source port</td><td>Destination port</td></tr>
<tr><td colspan="4">Sequence number</td></tr>
<tr><td colspan="4">Acknowledgment</td></tr>
<tr><td>HdrLen</td><td>0</td><td>Flags</td><td>Advertised window</td></tr>
<tr><td colspan="3">Checksum</td><td>Urgent pointer</td></tr>
<tr><td colspan="4">Options (variable)</td></tr>
<tr><td colspan="4">Data</td></tr>
</table>

TCP client lifecycle



TCP server lifecycle



提问

Q & A