NANJING UNIVERSITY

运输层

殷亚凤
智能软件与工程学院
苏州校区南雍楼东区225
yafeng@nju.edu.cn，https://yafengnju.github.io/

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)

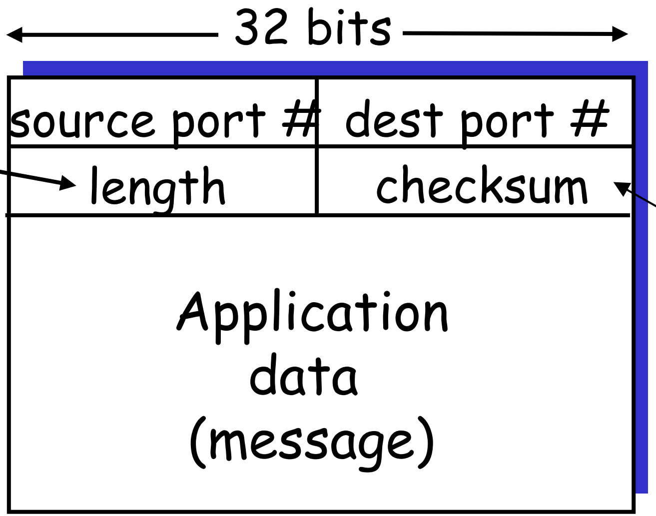
- "Best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- Connectionless:
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others
- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP

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

Length in octets, including Header and Data



Header + Data + Pseudo-header;
Or set to 0 if no check

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

Outline

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

The TCP Abstraction

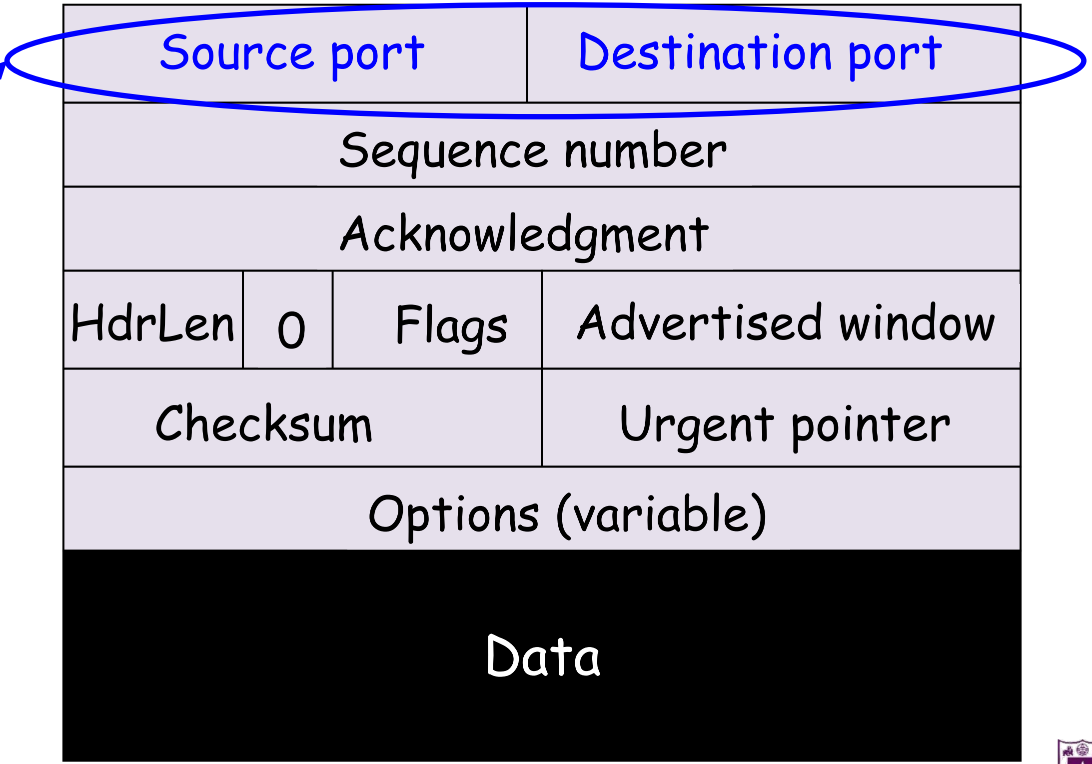
- TCP delivers a reliable, in-order, byte stream
- Reliable: TCP resends lost packets (recursively)
    - Until it gives up and shuts down connection
- In-order: TCP only hands consecutive chunks of data to application
- Byte stream: TCP assumes there is an incoming stream of data, and attempts to deliver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



NANJING UNIVERSITY

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?

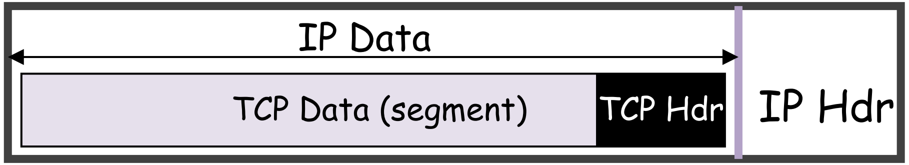
- Most of what we've seen
  - Checksum
  - Sequence numbers are byte offsets

TCP header

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

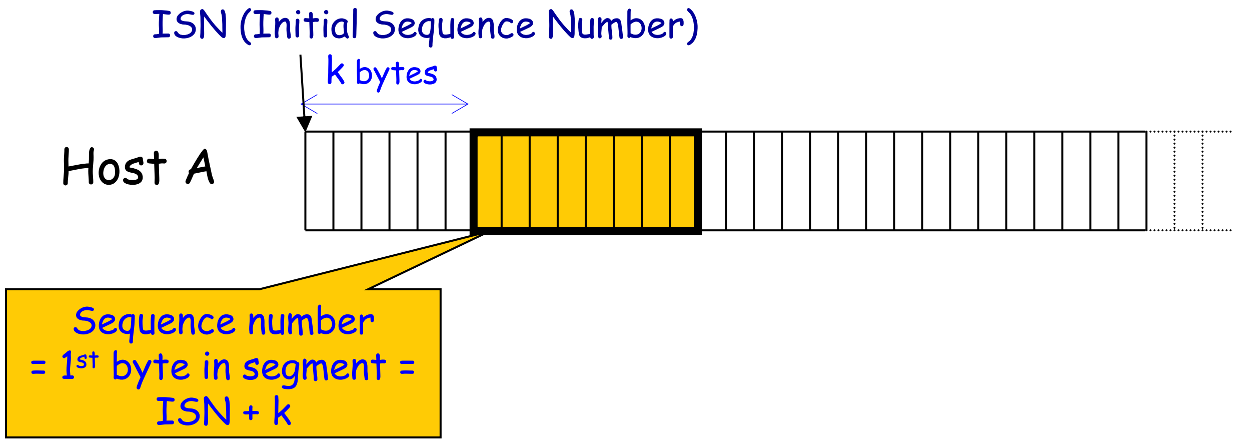
Source port		Destination port	
Sequence number			
Acknowledgment			
HdrLe	0	Flags	Advertised window
Checksum			Urgent pointer
Options (variable)			
Data			

TCP segment

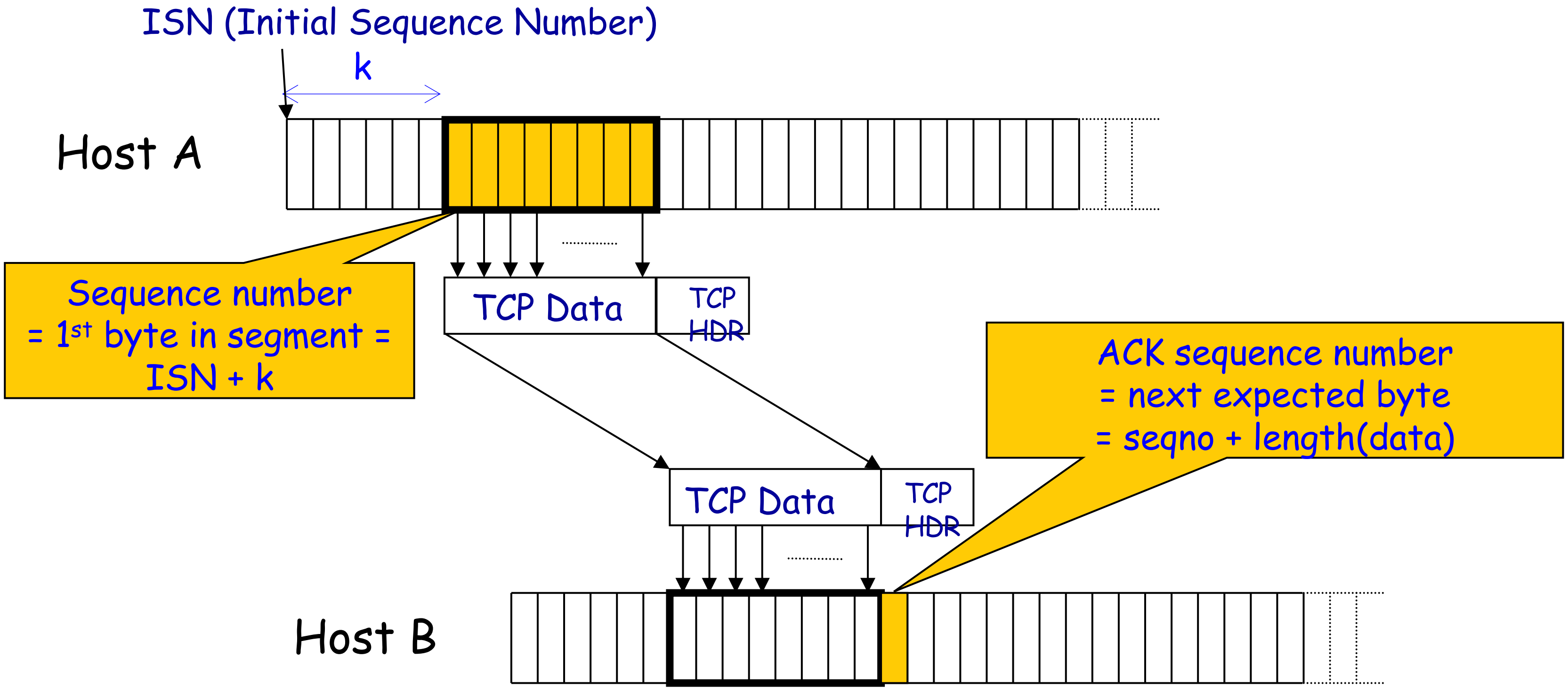


- IP packet
  - No bigger than Maximum Transmission Unit (MTU)
  - E.g., up to 1500 bytes with Ethernet
- TCP packet
  - IP packet with a TCP header and data inside
  - TCP header $\geq$ 20 bytes long
- TCP segment
  - No more than Maximum Segment Size (MSS) bytes
  - E.g., up to 1460 consecutive bytes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

Sequence numbers



Sequence numbers



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

南京大學 NANJING UNIVERSITY

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

TCP header

Acknowledgment gives seqno just beyond highest seqno received in order

<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)
  - 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?

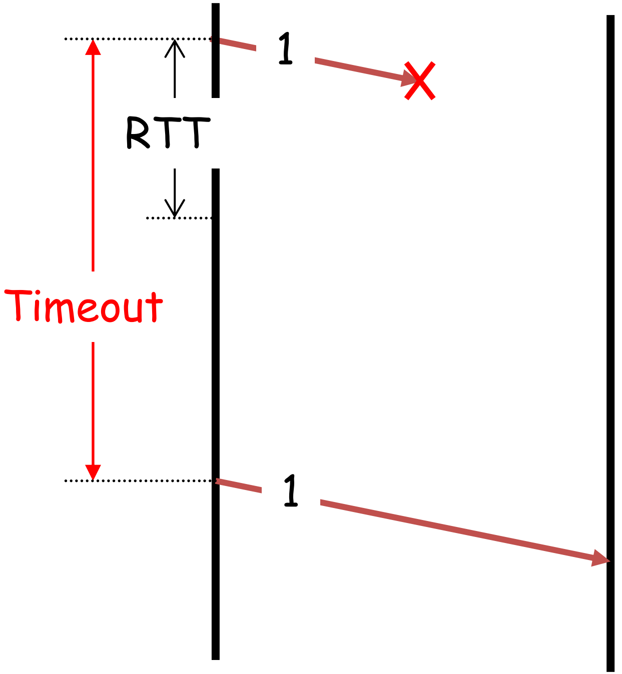
What does TCP introduce?

- 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

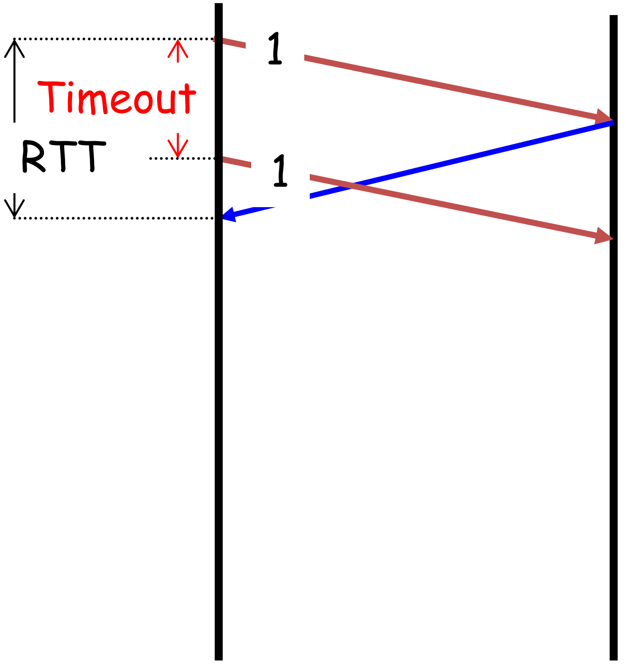
Retransmission timeout

- 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

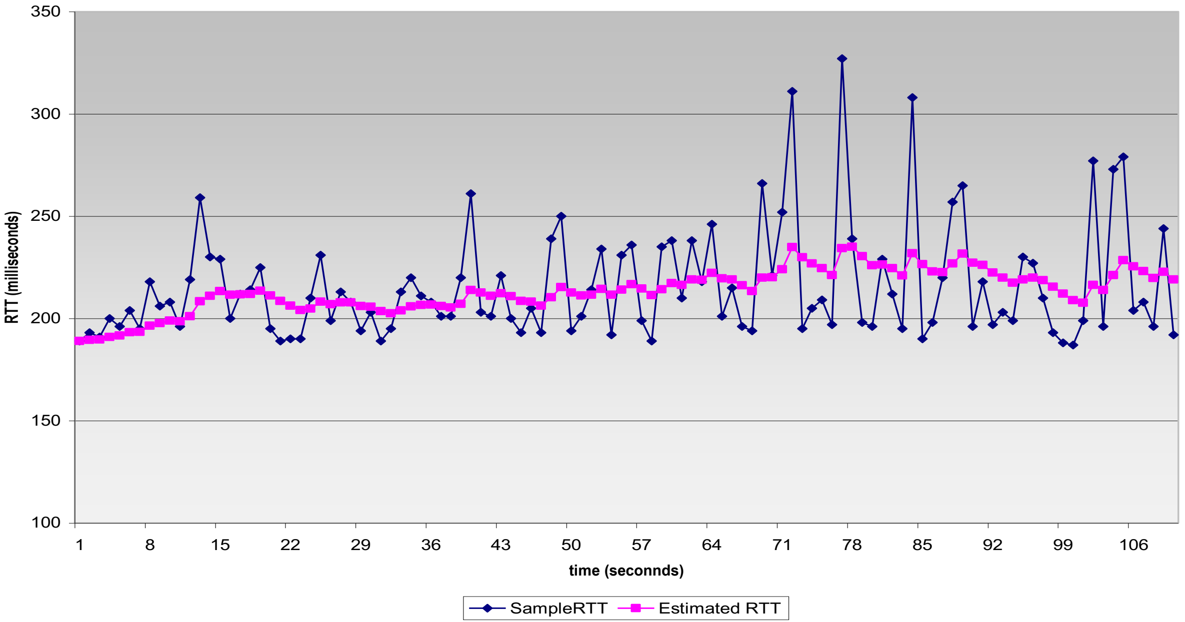
Retransmission timeout

- 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



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

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

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

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

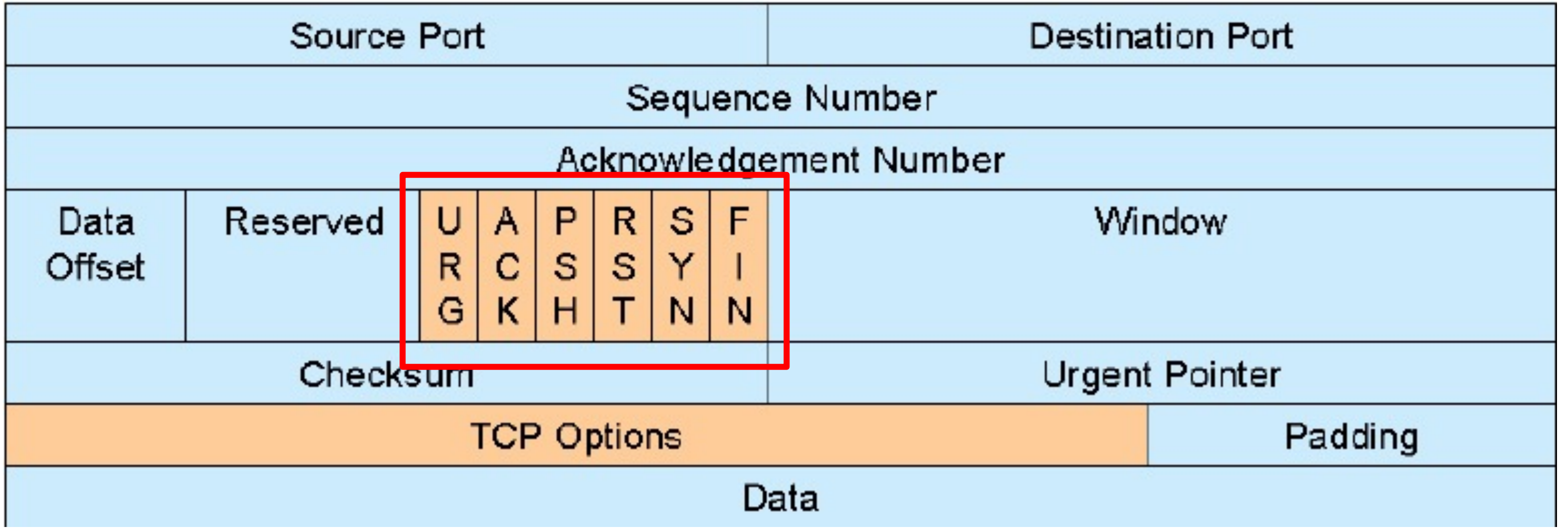
$$RTO(k+1) = SRTT(k+1) + f \times SDEV(k+1)$$

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

Outline

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

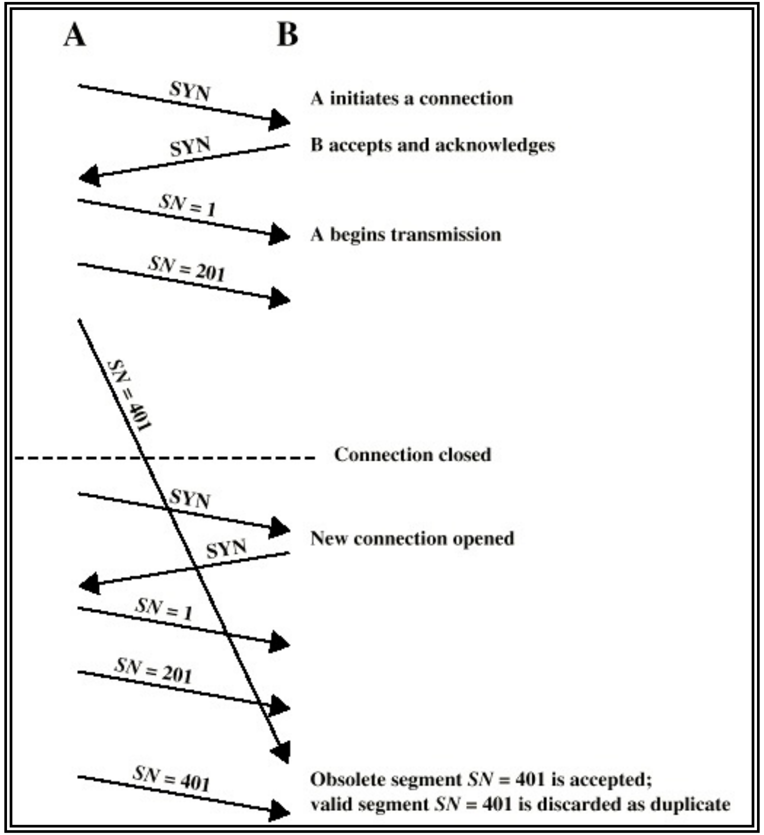
- TCP header field for connection establishment and teardown



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

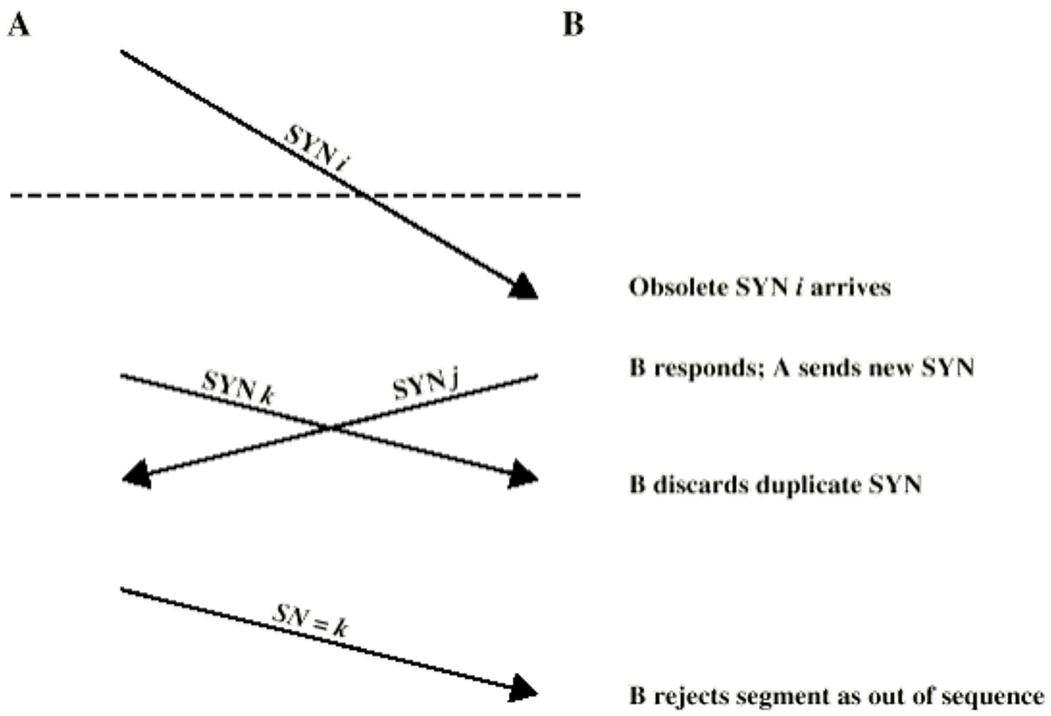


Initial Sequence Number (ISN)

- 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.

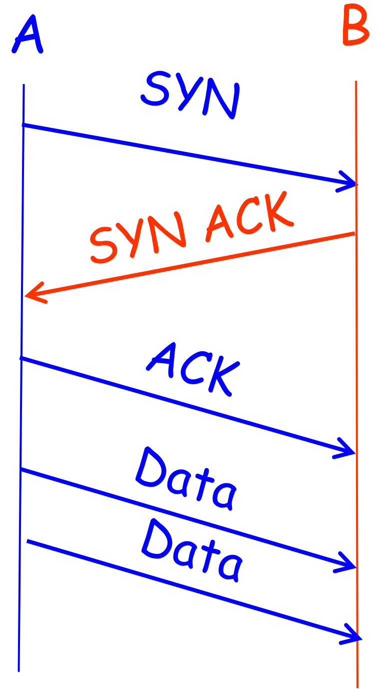
- However:

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>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>

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

B's port		A's port	
B's Initial Sequence Number			
ACK=A's ISN+1			
5	0	SYN|ACK	Advertised window
Checksum		Urgent pointer	

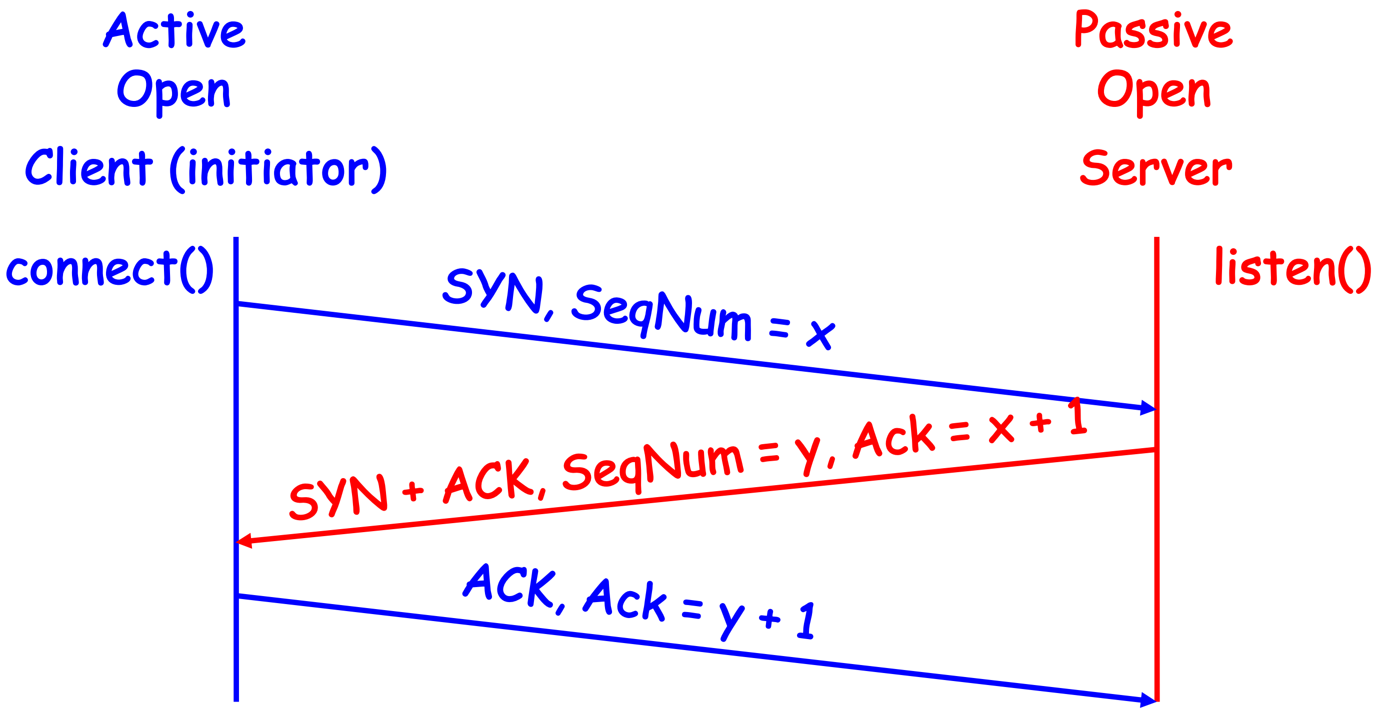
Step 1: A's ACK to SYN-ACK

A tells B to open a connection

A's port		B's port	
A's Initial Sequence Number			
ACK=B's ISN+1			
5	0	ACK	Advertised window
Checksum		Urgent pointer	

南京大學
NANJING UNIVERSITY

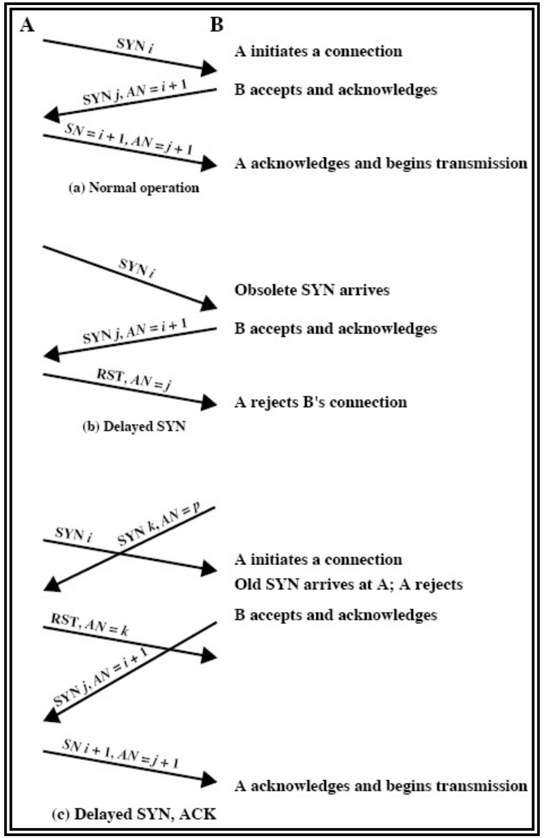
TCP's 3-Way handshaking



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

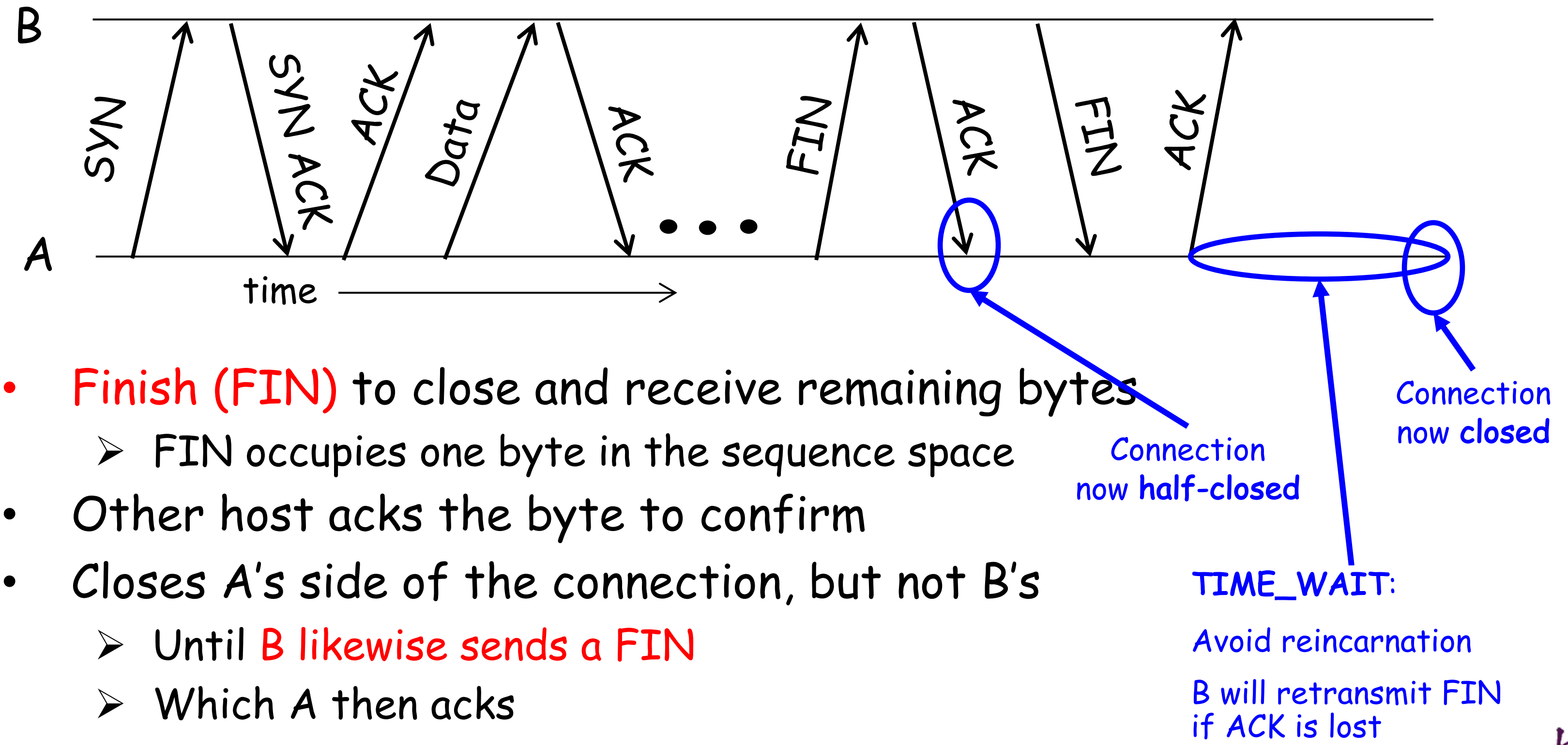
Three-Way Handshake: Examples



Outline

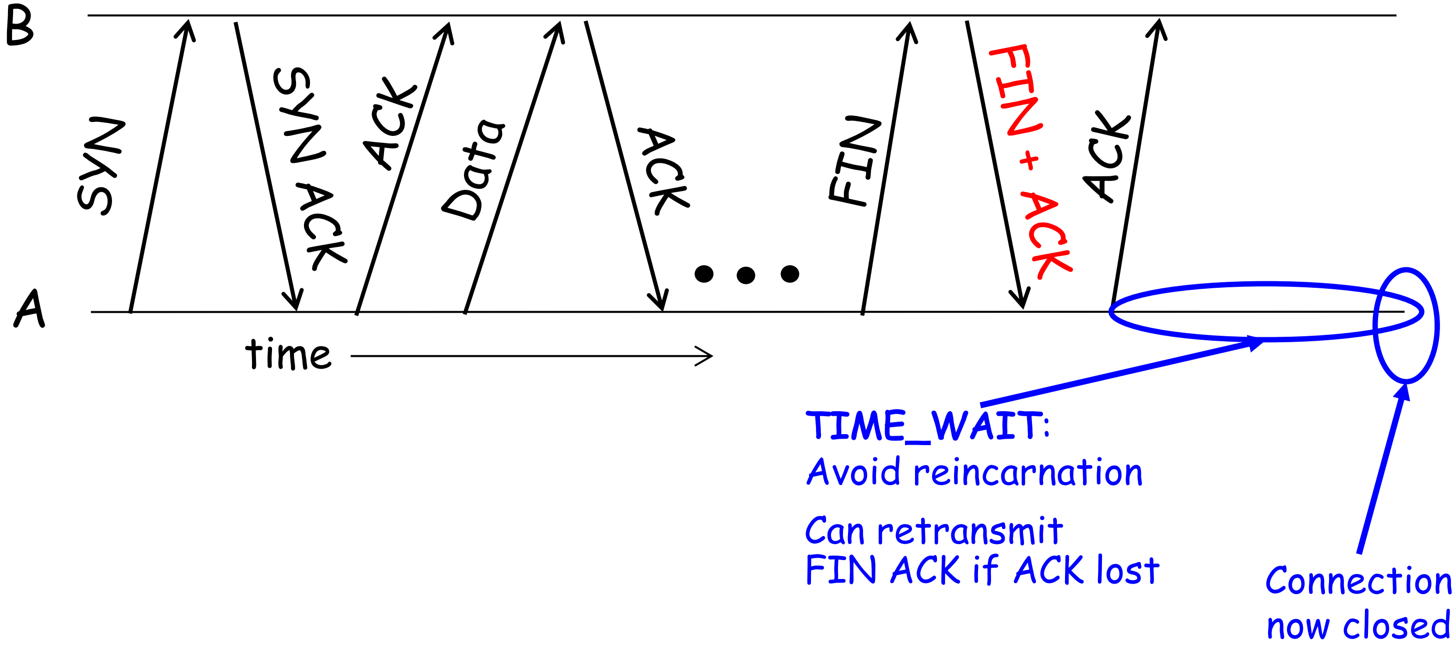
- UDP: User Datagram Protocol
- TCP: Transmission Control Protocol
- TCP Connection Setup
- TCP Connection Teardown

Normal termination, one side at a time



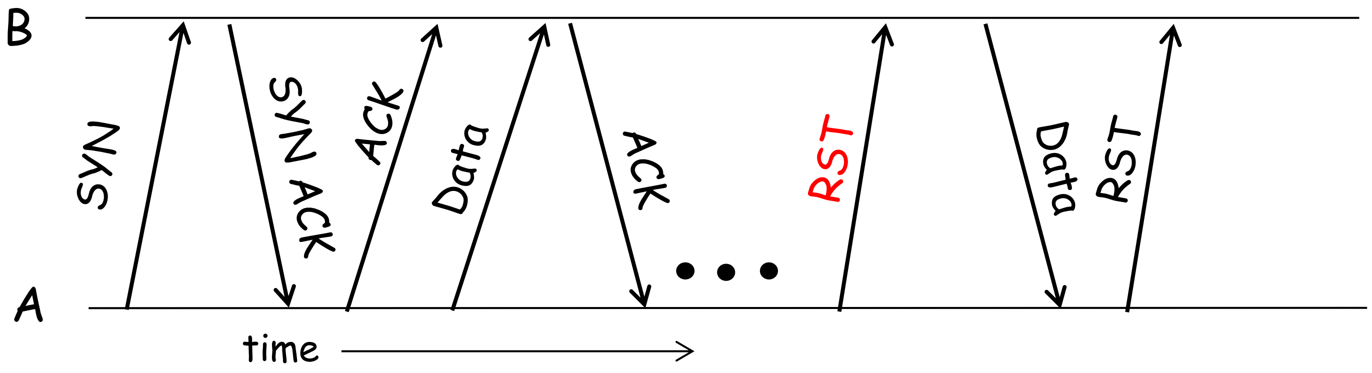
- **Finish (FIN)** to close and receive remaining bytes
  - FIN occupies one byte in the sequence space
- Other host acks the byte to confirm
- Closes A's side of the connection, but not B's
  - Until B likewise sends a FIN
  - Which A then acks

Normal termination, both together



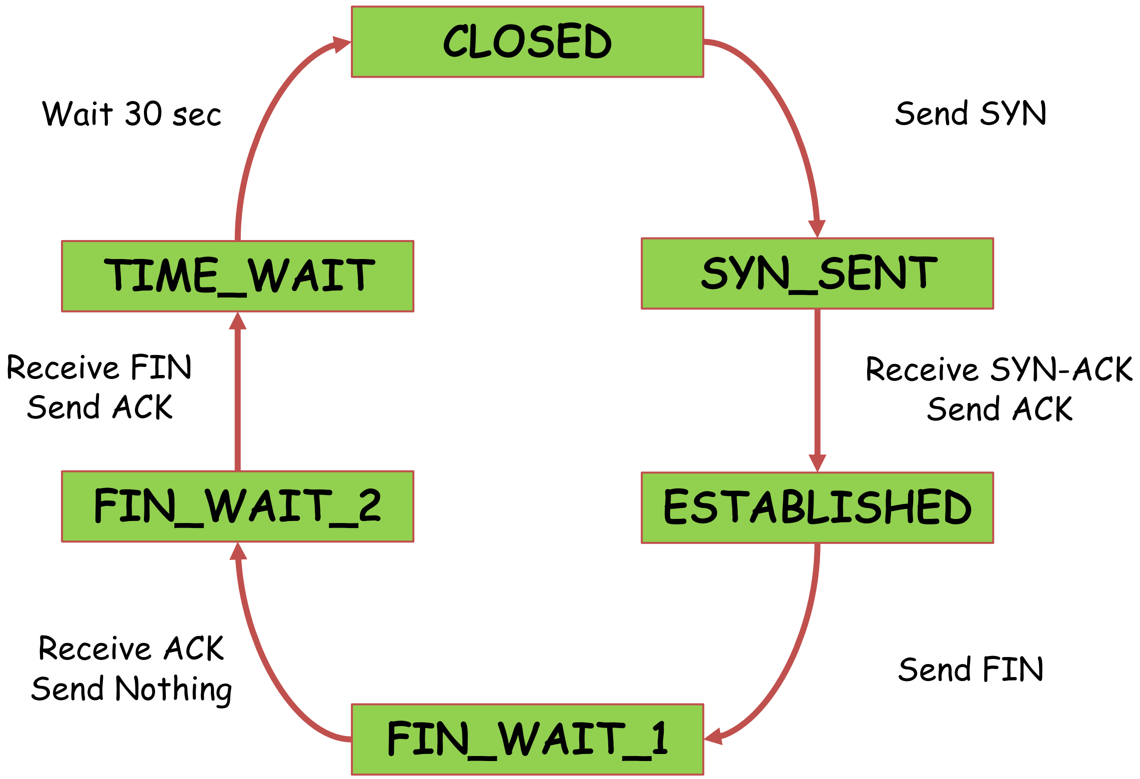
- Same as before, but B sets FIN with their ack of A's FIN

Abrupt termination

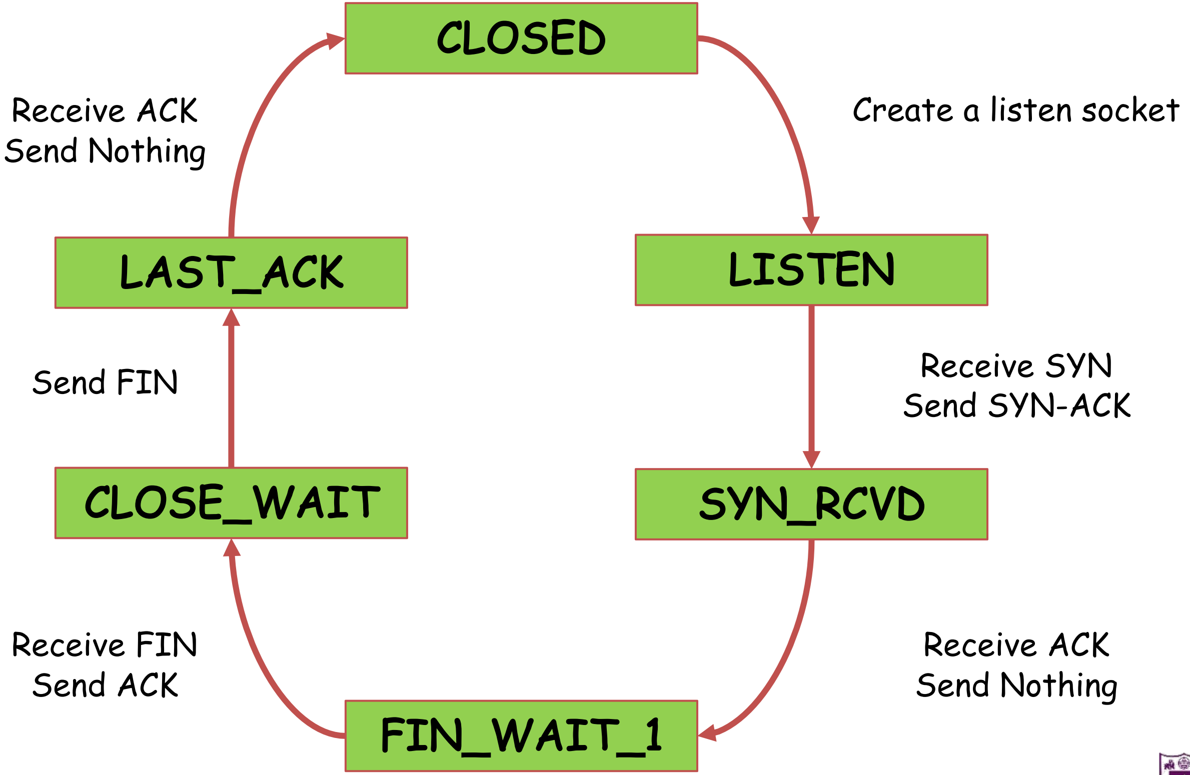


- A sends a RESET (RST) to B
  - E.g., because application process on A crashed
- That's it
  - B does not ack the RST
  - Thus, RST is not delivered reliably, and any data in flight is lost
  - But: if B sends anything more, will elicit another RST

TCP client lifecycle



TCP server lifecycle



课程习题（作业）——截止日期：4月8日晚23:59

- **课本187-195页**：第R5、R8、R14、P1、P3、P5、P19、P20题
- 提交方式：https://selearning.nju.edu.cn/（教学支持系统）

- 命名：学号+姓名+第*章。
- 若提交遇到问题请及时发邮件或在下一次上课时反馈。

南京大學
NANJING UNIVERSITY

课程习题（作业）——截止日期：4月8日晚23:59

R5. 在今天的因特网中，为什么语音和图像流量常常是经过 TCP 而不是经 UDP 发送。（提示：我们寻找的答案与 TCP 的拥塞控制机制没有关系。）

R8. 假定在主机 C 端口 80 上运行的一个 Web 服务器。假定这个 Web 服务器使用持续连接，并且正在接收来自两台不同主机 A 和 B 的请求。被发送的所有请求都通过位于主机 C 的相同套接字吗？如果它们通过不同的套接字传递，这两个套接字都具有端口 80 吗？讨论和解释之。

R14. 是非判断题：

a. 主机 A 经过一条 TCP 连接向主机 B 发送一个大文件。假设主机 B 没有数据发往主机 A。因为主机 B 不能随数据捎带确认，所以主机 B 将不向主机 A 发送确认。

b. 在连接的整个过程中，TCP 的 rwnd 的长度决不会变化。

c. 假设主机 A 通过一条 TCP 连接向主机 B 发送一个大文件。主机 A 发送但未被确认的字节数不会超过接收缓存的大小。

d. 假设主机 A 通过一条 TCP 连接向主机 B 发送一个大文件。如果对于这条连接的一个报文段的序号为 m，则对于后继报文段的序号将必然是 $m+1$。

e. TCP 报文段在它的首部中有一个 rwnd 字段。

f. 假定在一条 TCP 连接中最后的 SampleRTT 等于 1 秒，那么对于该连接的 TimeoutInterval 的当前值必定大于等于 1 秒。

g. 假设主机 A 通过一条 TCP 连接向主机 B 发送一个序号为 38 的 4 个字节的报文段。在这个相同的报文段中，确认号必定是 42。

课程习题（作业）——截止日期：4月8日晚23:59

P1. 假设客户 A 向服务器 S 发起一个 Telnet 会话。与此同时，客户 B 也向服务器 S 发起一个 Telnet 会话。给出下面报文段的源端口号和目的端口号：

a. 从 A 向 S 发送的报文段。

b. 从 B 向 S 发送的报文段。

c. 从 S 向 A 发送的报文段。

d. 从 S 向 B 发送的报文段。

e. 如果 A 和 B 是不同的主机，那么从 A 向 S 发送的报文段的源端口号是否可能与从 B 向 S 发送的报文段的源端口号相同？

f. 如果它们是同一台主机，情况会怎么样？

P3. UDP 和 TCP 使用反码来计算它们的检验和。假设你有下面 3 个 8 比特字节：01010011，01100110，01110100。这些 8 比特字节和的反码是多少？（注意到尽管 UDP 和 TCP 使用 16 比特的字来计算检验和，但对于这个问题，你应该考虑 8 比特和。）写出所有工作过程。UDP 为什么要用该和的反码，即为什么不直接使用该和呢？使用该反码方案，接收方如何检测出差错？1 比特的差错将可能检测不出来吗？2 比特的差错呢？

课程习题（作业）——截止日期：4月8日晚23:59

P5. 假定某 UDP 接收方对接收到的 UDP 报文段计算因特网检验和，并发现它与承载在检验和字段中的值相匹配。该接收方能够绝对确信没有出现过比特差错吗？试解释之。

P19. 考虑一种情况，主机 A 想同时向主机 B 和主机 C 发送分组。A 与 B 和 C 是经过广播信道连接的，即由 A 发送的分组通过该信道传送到 B 和 C。假设连接 A、B 和 C 的这个广播信道具有独立的报文丢失和损坏特性（例如，从 A 发出的报文可能被 B 正确接收，但没有被 C 正确接收）。设计一个类似于停等协议的差错控制协议，用于从 A 可靠地传输分组到 B 和 C。该协议使得 A 直到得知 B 和 C 已经正确接收到当前报文，才获取上层交付的新数据。给出 A 和 C 的 FSM 描述。（提示：B 的 FSM 大体上应当与 C 的相同。）同时，给出所使用的报文格式的描述。

P20. 考虑一种主机 A 和主机 B 要向主机 C 发送报文的情况。主机 A 和 C 通过一条报文能够丢失和损坏（但不重排序）的信道相连接。主机 B 和 C 由另一条（与连接 A 和 C 的信道独立）具有相同性质的信道连接。在主机 C 上的运输层，在向上层交付来自主机 A 和 B 的报文时应当交替进行（即它应当首先交付来自 A 的分组中的数据，然后是来自 B 的分组中的数据，等等）。设计一个类似于停等协议的差错控制协议，以可靠地向 C 传输来自 A 和 B 的分组，同时以前面描述的方式在 C 处交替地交付。给出 A 和 C 的 FSM 描述。（提示：B 的 FSM 大体上应当与 A 的相同。）同时，给出所使用的报文格式的描述。

提问

Q & A

殷亚凤
智能软件与工程学院
苏州校区南雍楼东区225
yafeng@nju.edu.cn，https://yafengnju.github.io/