CS 181AG
Lecture 18

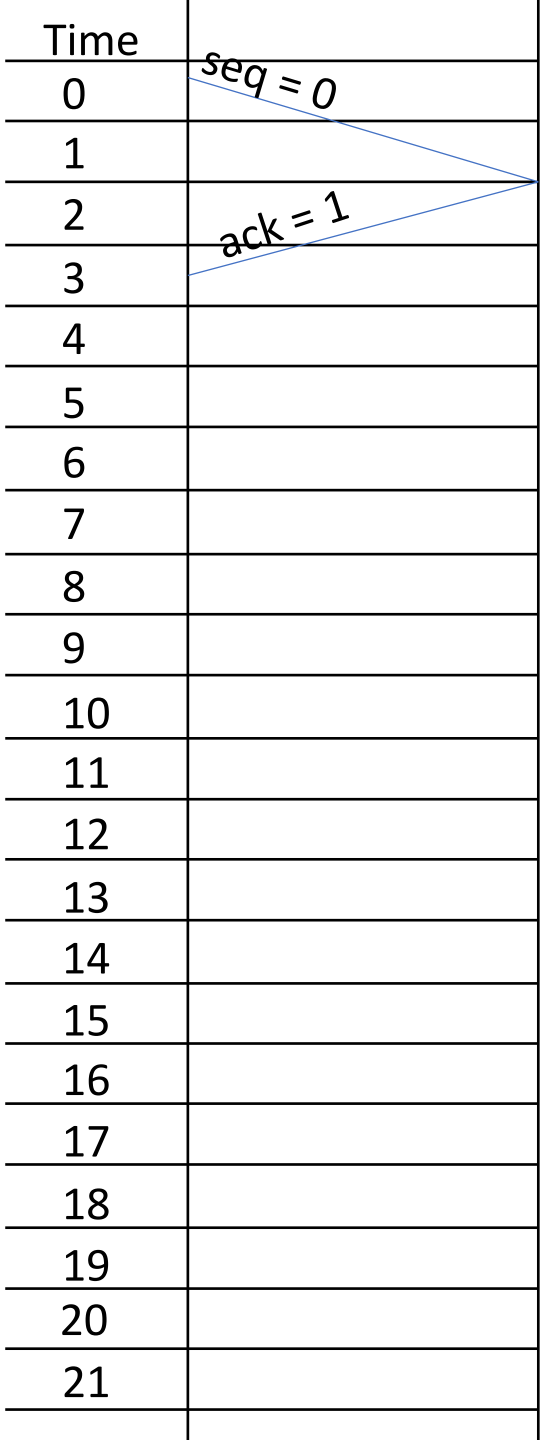
TCP

Arthi Padmanabhan

Nov 7, 2022

Assignment Clarifications/Hints

- Problem 1:
  - Requests are the same for the first iteration of each round, not between iterations. You can still just draw grant and accept, but people have found it helpful to draw all three for each iteration – you may do that if it helps you
- Problem 3:
  - See note about ack=1
  - Assume 1 ack per data packet
  - I’ve shown the first packet getting sent and an ack being received. That doesn’t mean the sender can’t send other packets before receiving that first ack (since size=1 and window size = 3)


Time
0
seq = 0
1
2
ack = 1
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

Announcements

- No office hours this Friday
- Assignment 8 will not be the typical problem set and will be modified such that not having access to me is fair to you

Recap

- TCP is responsible for chopping up data into packets, sending them across the network, and dealing with out of order arrivals and lost packets
- Lost packets happen because routers’ buffers fill up
- Different TCP connections are identified using port numbers
- Selective Repeat ARQ (Automatic Repeat Request) uses sequence numbers and acks to determine which packet(s) to resend

TCP Header

- Src port
- Dst port
- Seq number
- Ack number
- Header length
- Reserved bits

TCP Header (cont.)

- 9 flags:
  - Nonce, CWR, ECN-echo -> used for congestion control
  - Urgent – not used anymore
  - Ack: this packet contains valid ack info
  - Psh: immediately push to app (don’t buffer)
  - Rst: abort connection; abnormal condition
  - Syn: used for connection setup
  - Fin: used for connection tear down

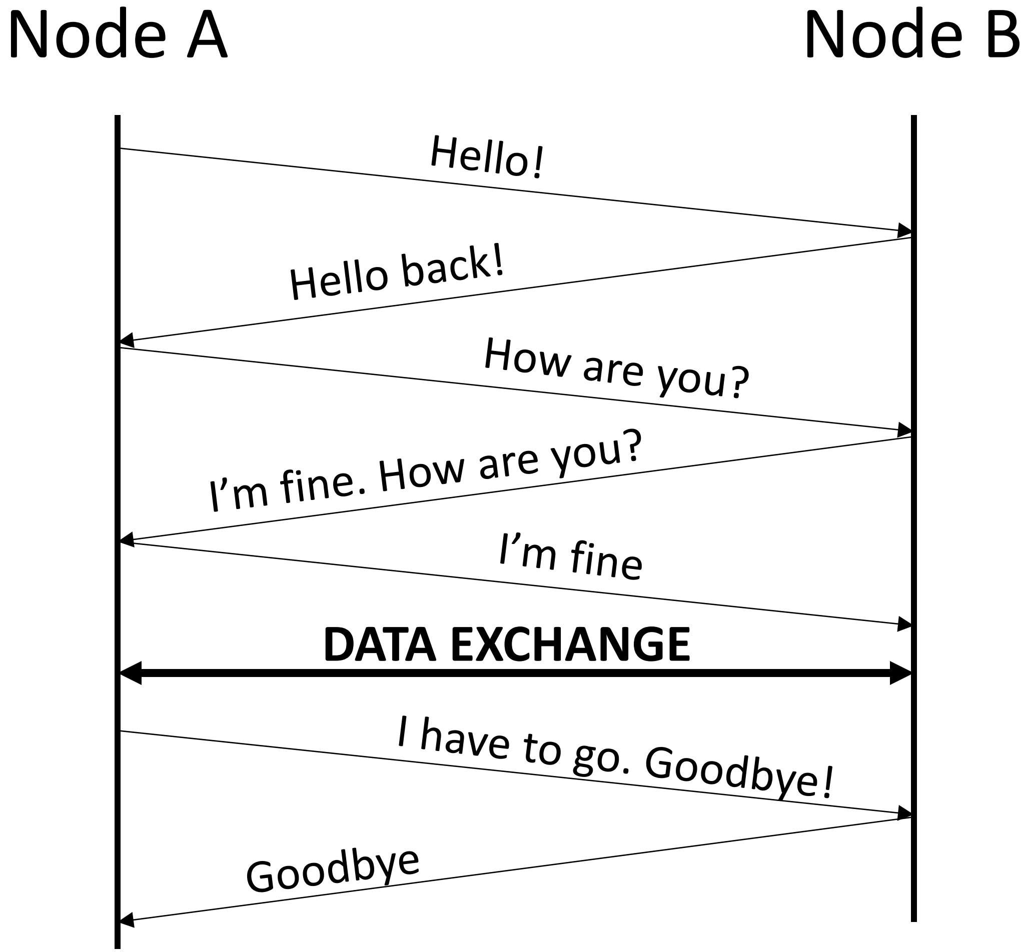
TCP Header (cont.)

- Window size
- Checksum
- Optional:
  - Max packet size
  - Window scale

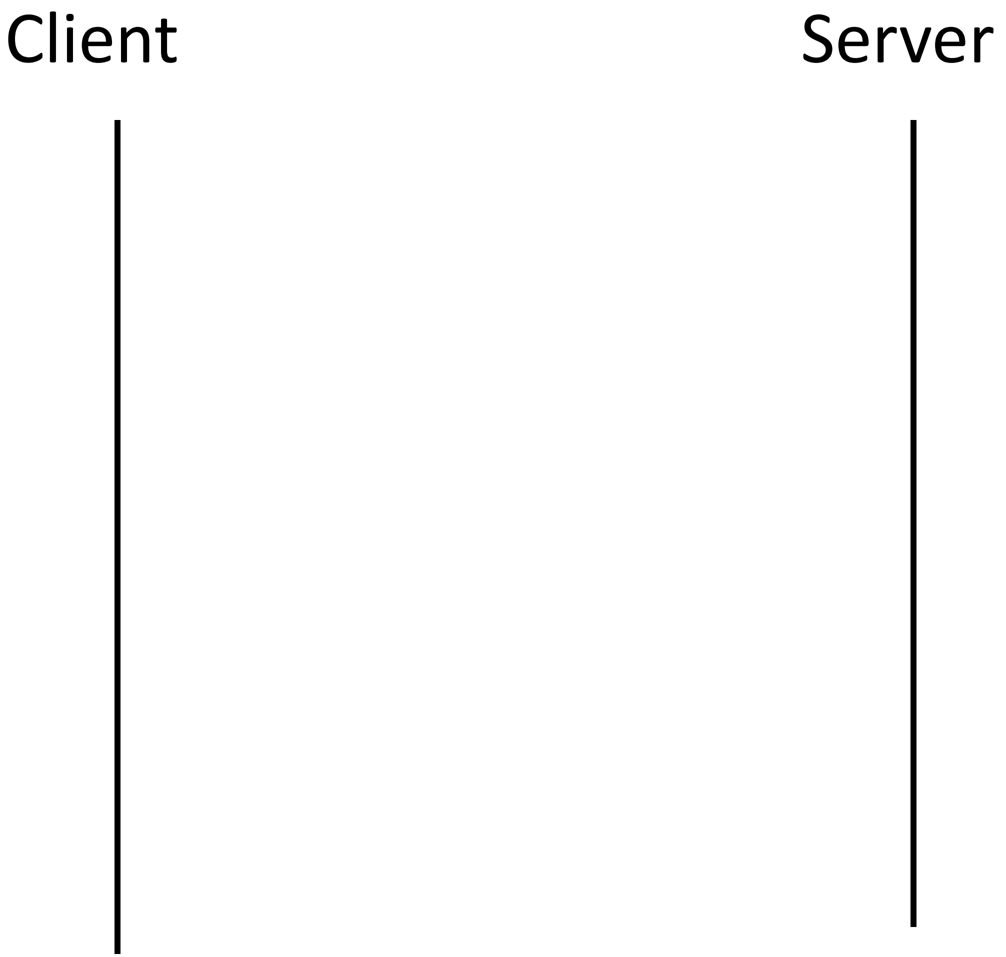
UDP Header

- UDP is for simple communication that needs speed (it is preferrable that a packet gets lost than retransmitted)
- Src port
- Dst port
- Length (total length of packet)
- Checksum

Starting a TCP Session

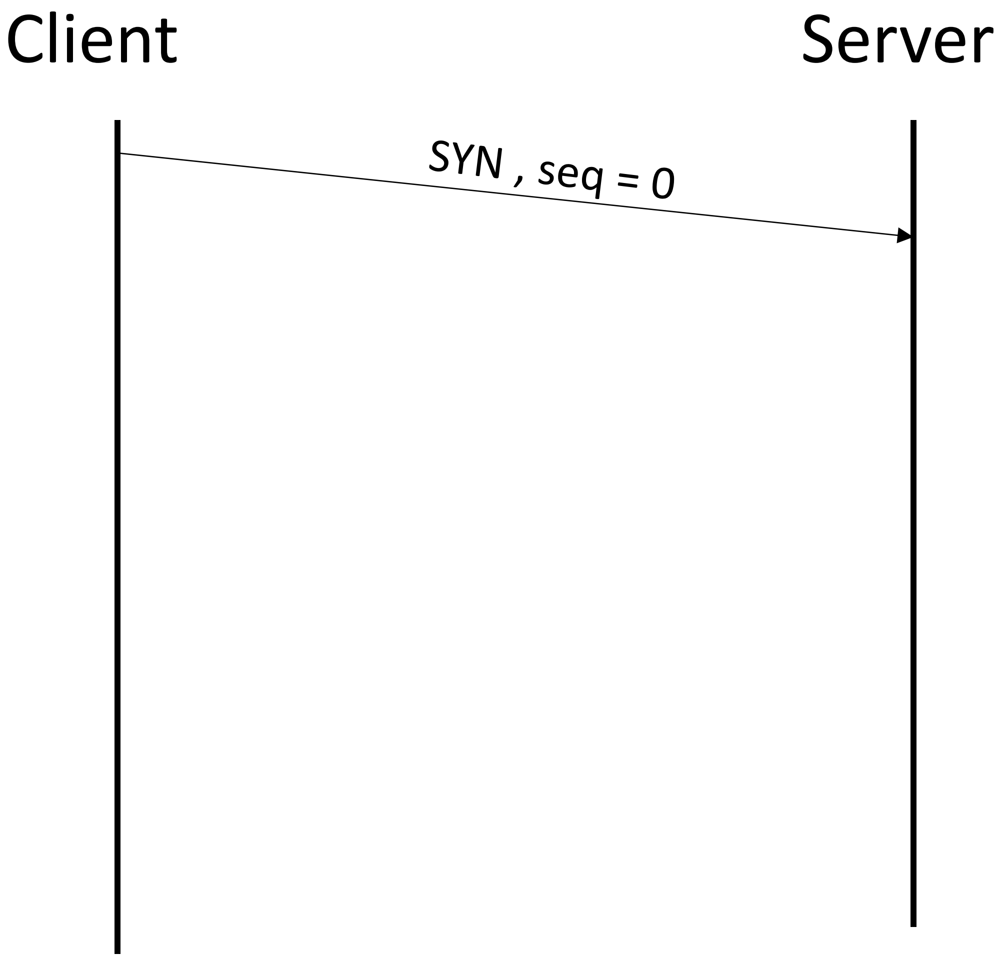


Starting a TCP Session (3-way handshake)



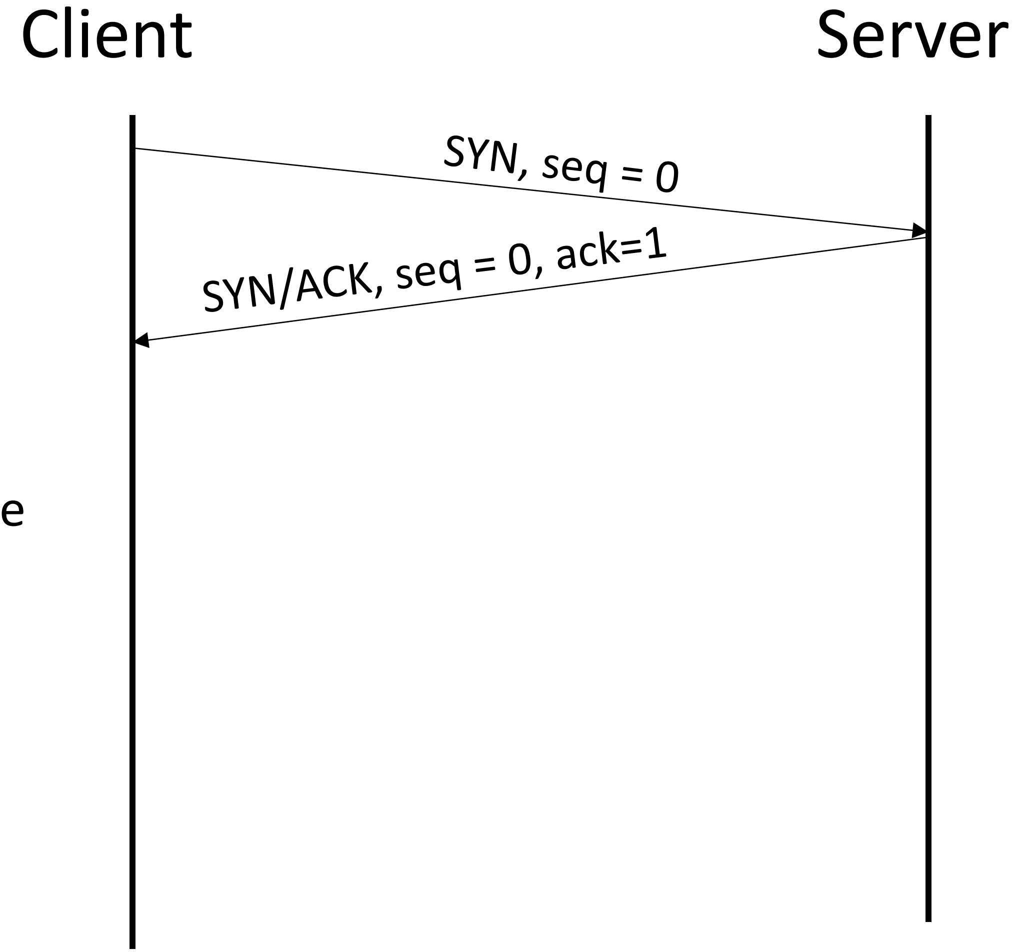
1) Server creates a connection and binds to port/address

Starting a TCP Session (3-way handshake)



1) Server creates a connection and binds to port/address
2) Client initiates a connection by sending TCP packet to server with SYN flag and initial seq number

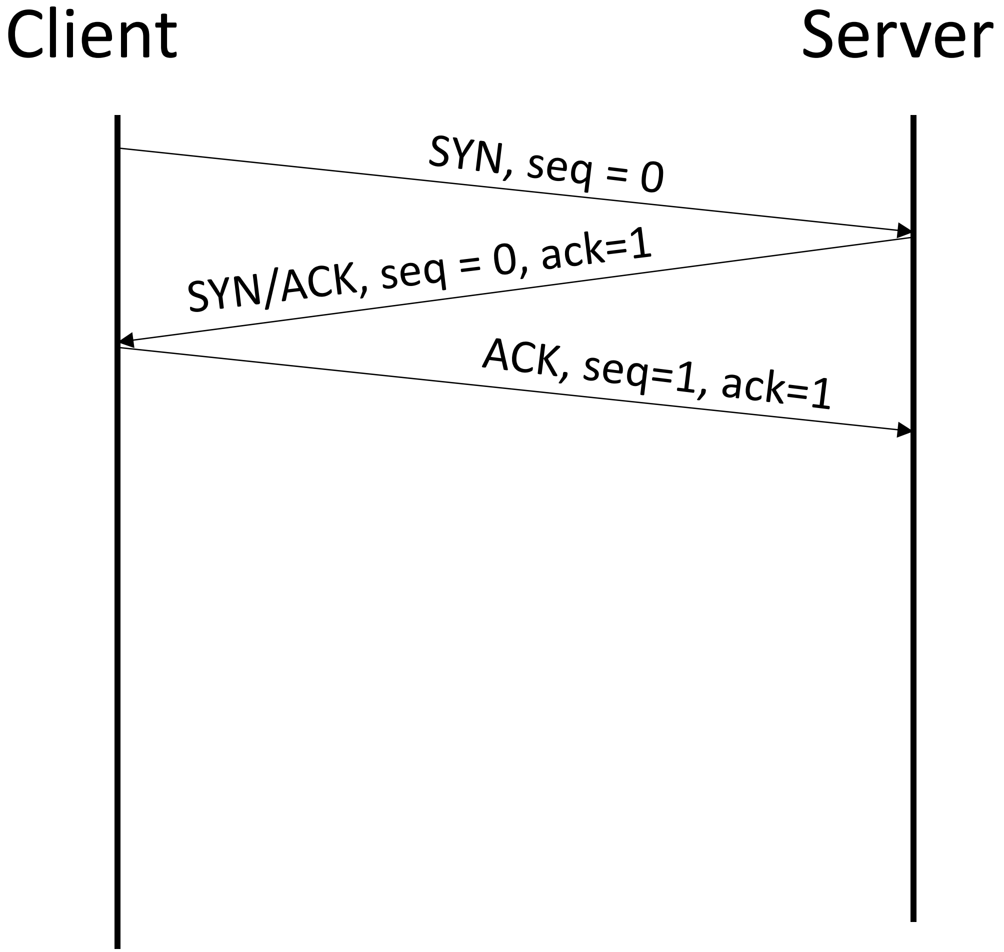
Starting a TCP Session (3-way handshake)



By convention, SYN packets have size 1 byte

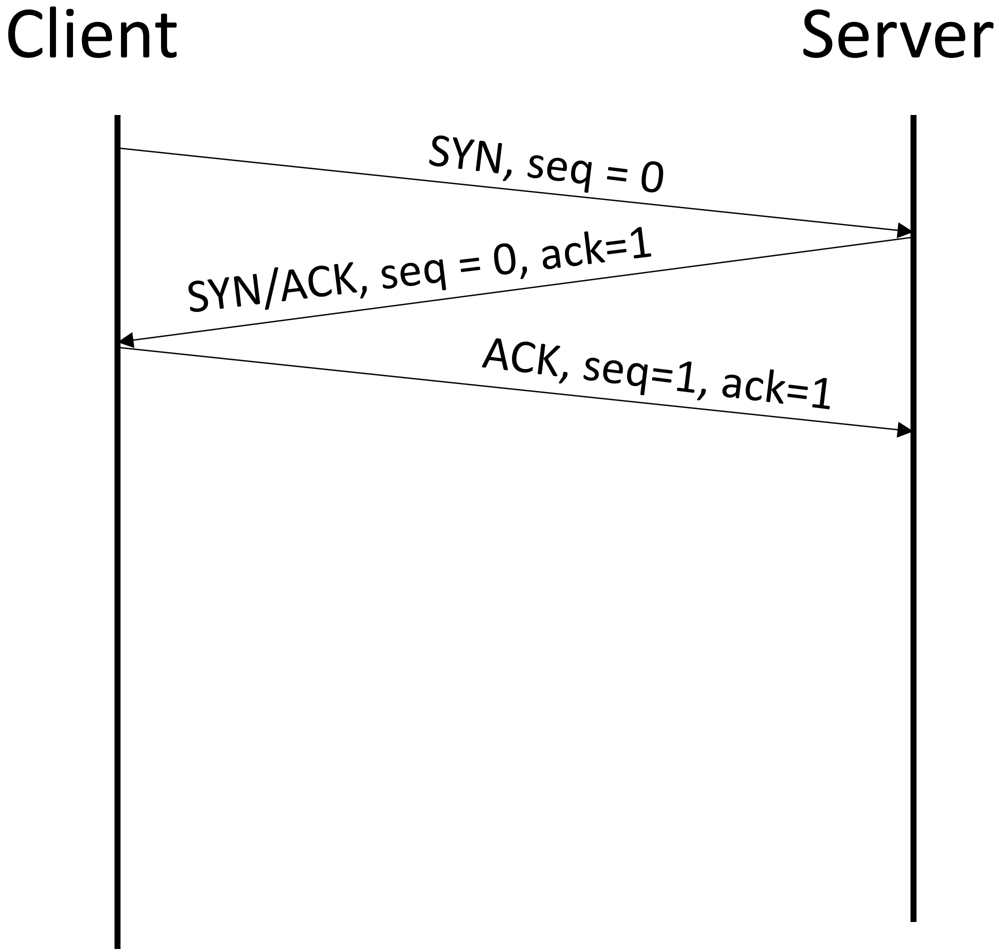
1) Server creates a connection and binds to port/address
2) Client initiates a connection by sending TCP packet to server with SYN flag and initial seq number
3) Server receives packet, sends back SYN/ACK with its own initial seq number and ack for clients initial seq number

Starting a TCP Session (3-way handshake)



4) Client receives SYN/ACK and sends back packet with ack set

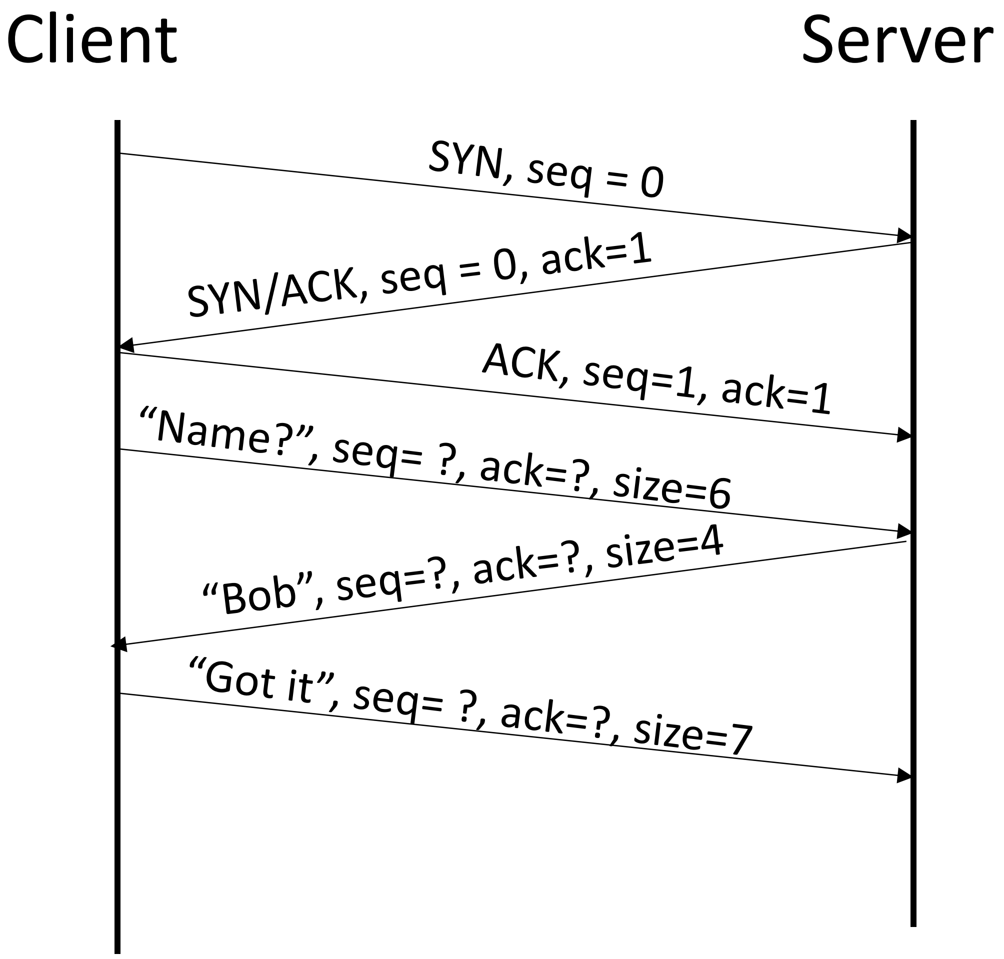
Starting a TCP Session (3-way handshake)



4) Client receives SYN/ACK and sends back packet with ack set

5) Server receives the ACK packet. Connection is established

Data Exchange


Client
Server
SYN, seq = 0
SYN/ACK, seq = 0, ack=1
ACK, seq=1, ack=1
“Name?”, seq= ?, ack=?, size=6
“Bob”, seq=?, ack=?, size=4
“Got it”, seq= ?, ack=?, size=7

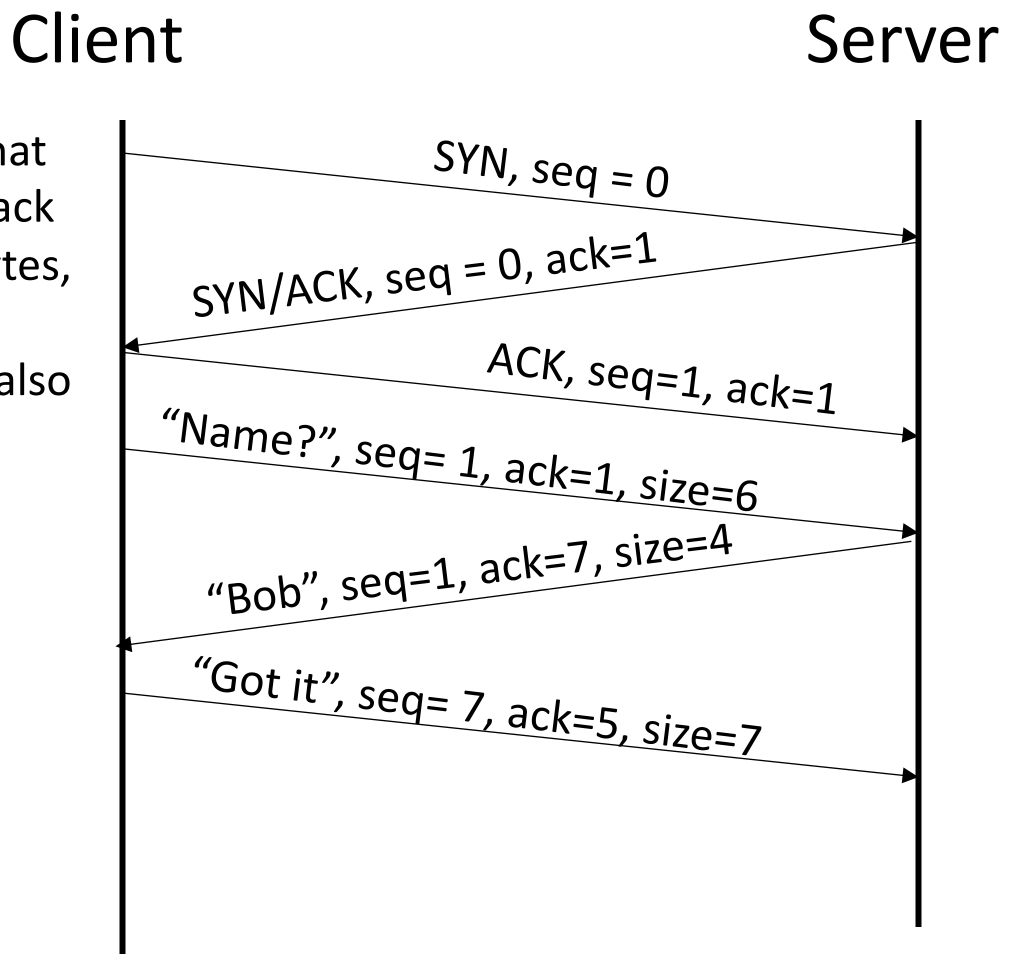
4) Client receives SYN/ACK and sends back packet with ack set

5) Server receives the ACK packet. Connection is established

6) Data exchange

Data Exchange

Note: a packet that just contains an ack has a size of 0 bytes, hence the next packet after ack also has seq = 1

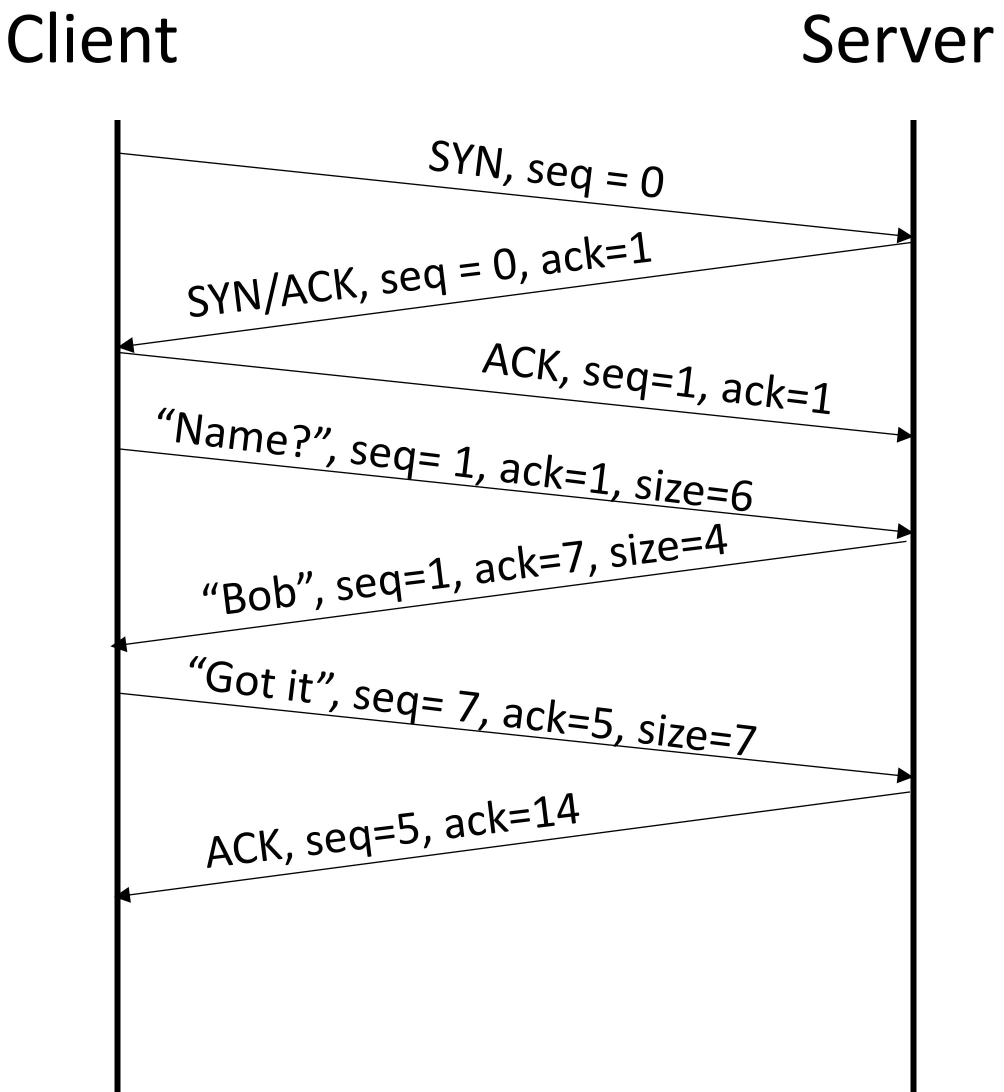

Client
Server
SYN, seq = 0
SYN/ACK, seq = 0, ack=1
ACK, seq=1, ack=1
“Name?”, seq= 1, ack=1, size=6
“Bob”, seq=1, ack=7, size=4
“Got it”, seq= 7, ack=5, size=7

4) Client receives SYN/ACK and sends back packet with ack set

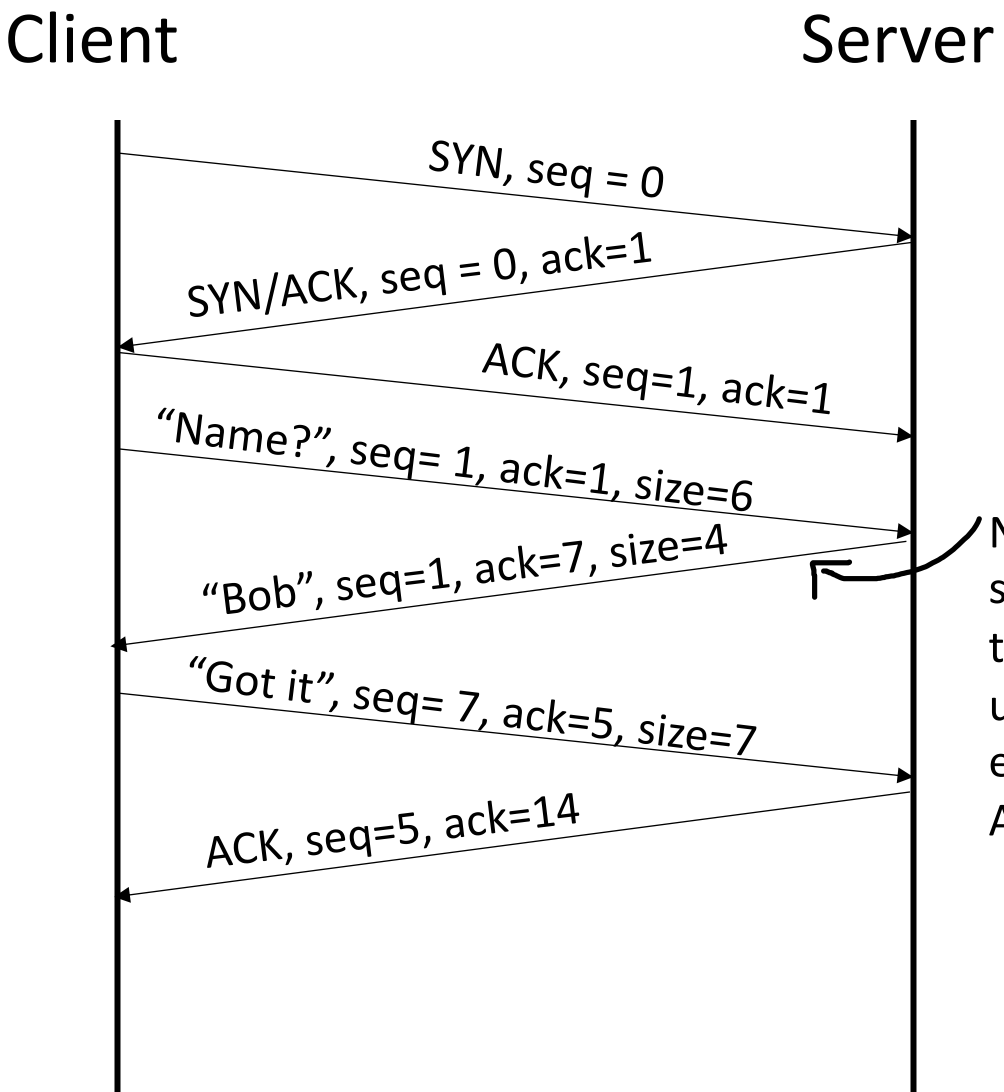
5) Server receives the ACK packet. Connection is established

6) Data exchange

Data Exchange

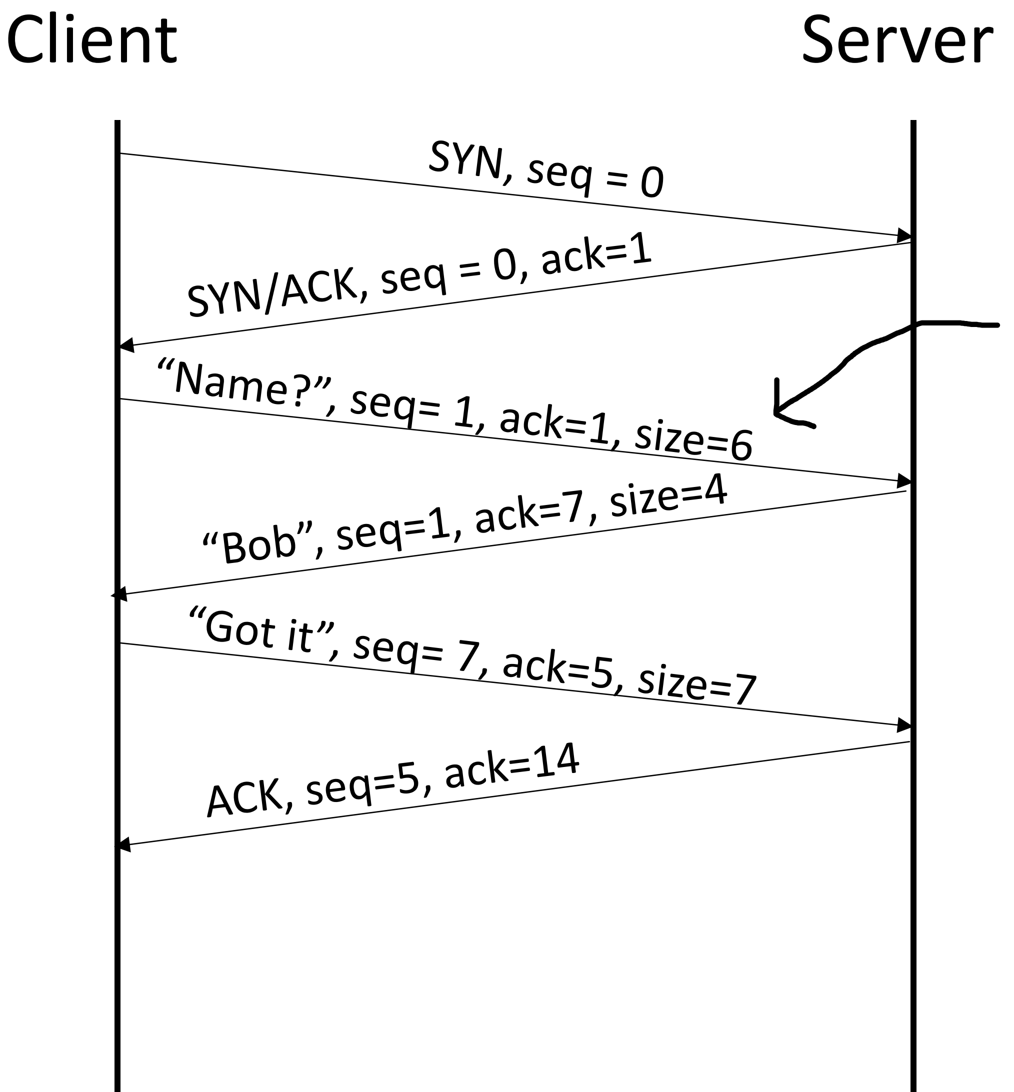

Client
Server
SYN, seq = 0
SYN/ACK, seq = 0, ack=1
ACK, seq=1, ack=1
“Name?”, seq= 1, ack=1, size=6
“Bob”, seq=1, ack=7, size=4
“Got it”, seq= 7, ack=5, size=7
ACK, seq=5, ack=14

Data Exchange


Client
Server
SYN, seq = 0
SYN/ACK, seq = 0, ack=1
ACK, seq=1, ack=1
“Name?”, seq= 1, ack=1, size=6
“Bob”, seq=1, ack=7, size=4
“Got it”, seq= 7, ack=5, size=7
ACK, seq=5, ack=14

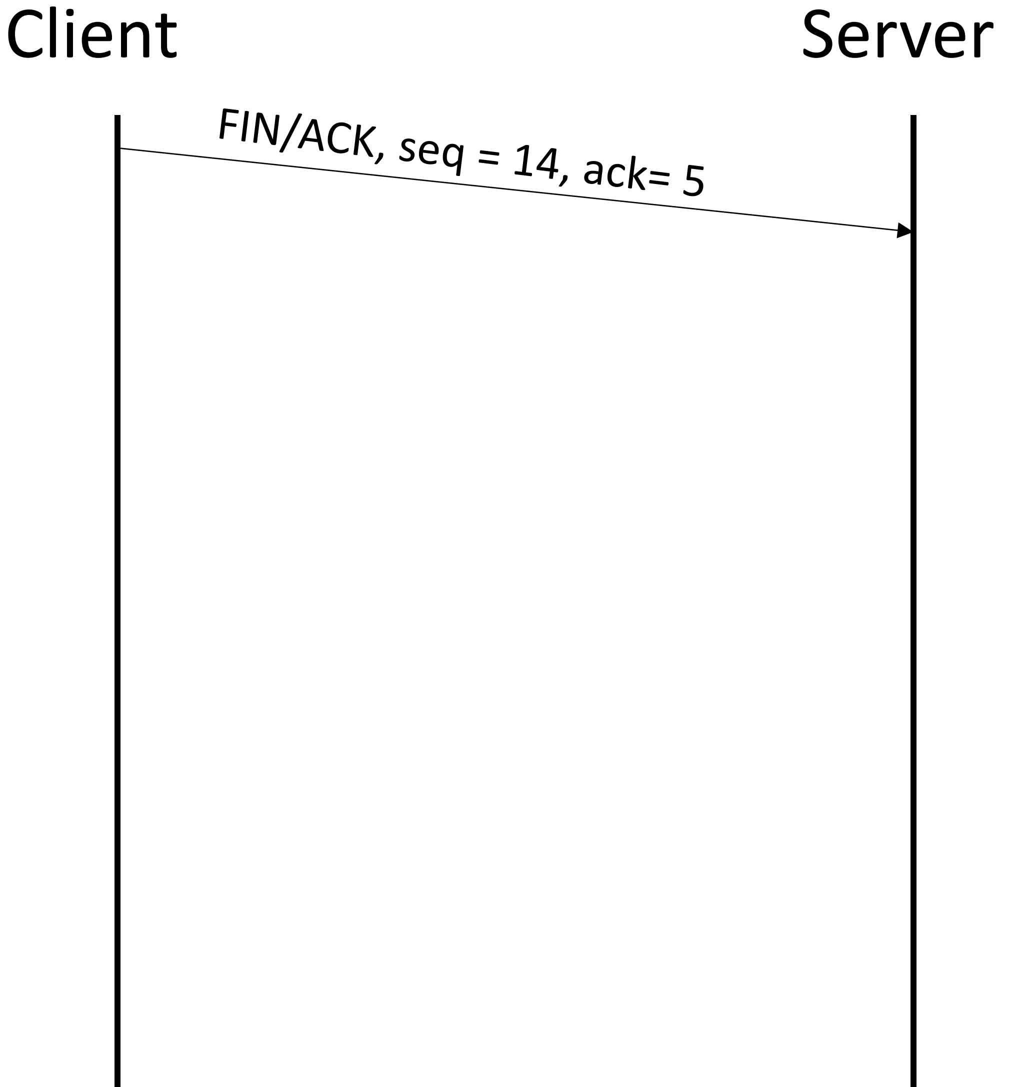
Notice that this packet both sends data and acknowledges the receipt of data. ACKs are usually “piggybacked”. Where else in this exchange could an ACK have been piggybacked?

Data Exchange


Client
Server
SYN, seq = 0
SYN/ACK, seq = 0, ack=1
“Name?”, seq= 1, ack=1, size=6
“Bob”, seq=1, ack=7, size=4
“Got it”, seq= 7, ack=5, size=7
ACK, seq=5, ack=14

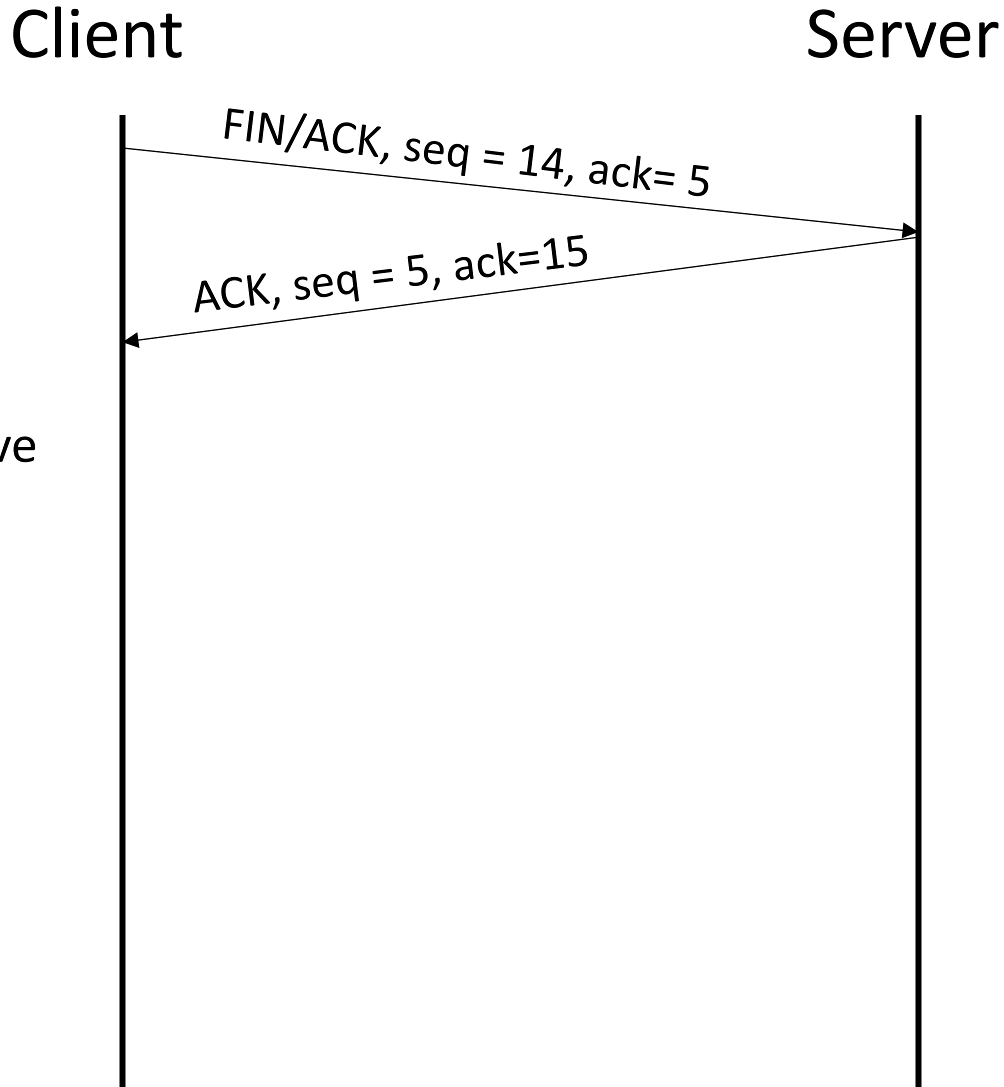
The 3-way handshake requires an ack here, but this could also be piggybacked with data. This packet both sends data and acknowledges receipt of the SYN-ACK

Connection Tear Down



7) Client sends packet to server with FIN/ACK flags set

Connection Tear Down

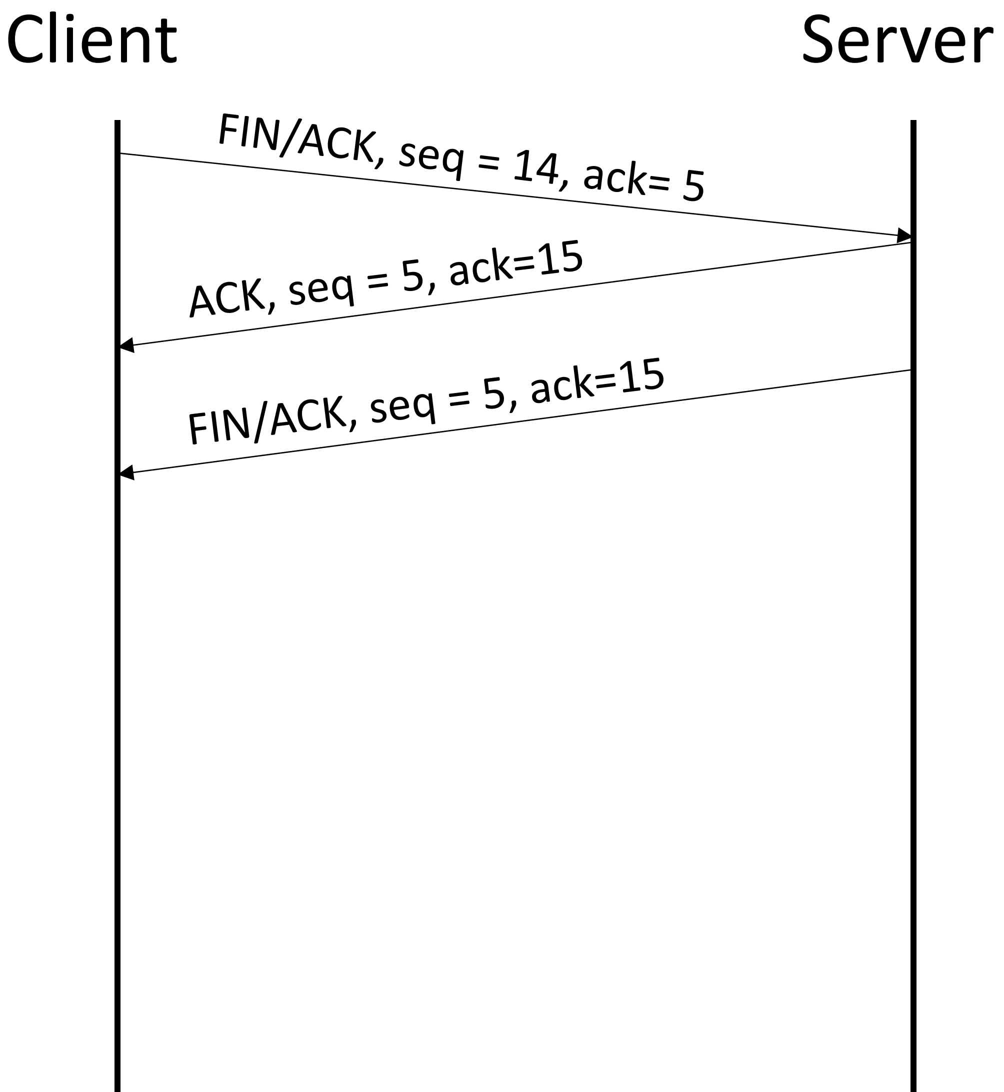


By convention, FIN packets have size 1 byte

7) Client sends packet to server with FIN/ACK flags set

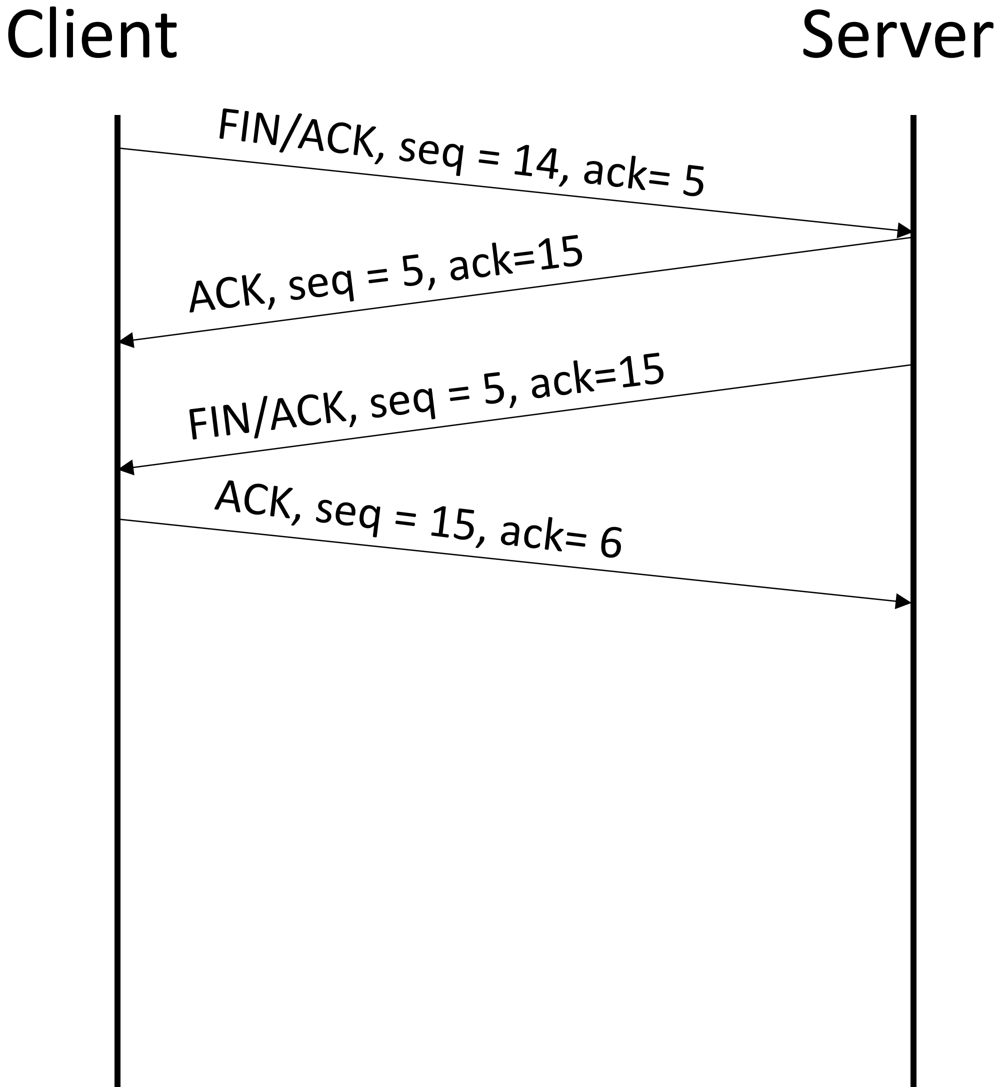
8) Server receives FIN packet and responds with ACK

Connection Tear Down



7) Client sends packet to server with FIN/ACK flags set
8) Server receives FIN packet and responds with ACK
9) Client receives ACK, says nothing
10) Server sends FIN,ACK

Connection Tear Down



7) Client sends packet to server with FIN/ACK flags set

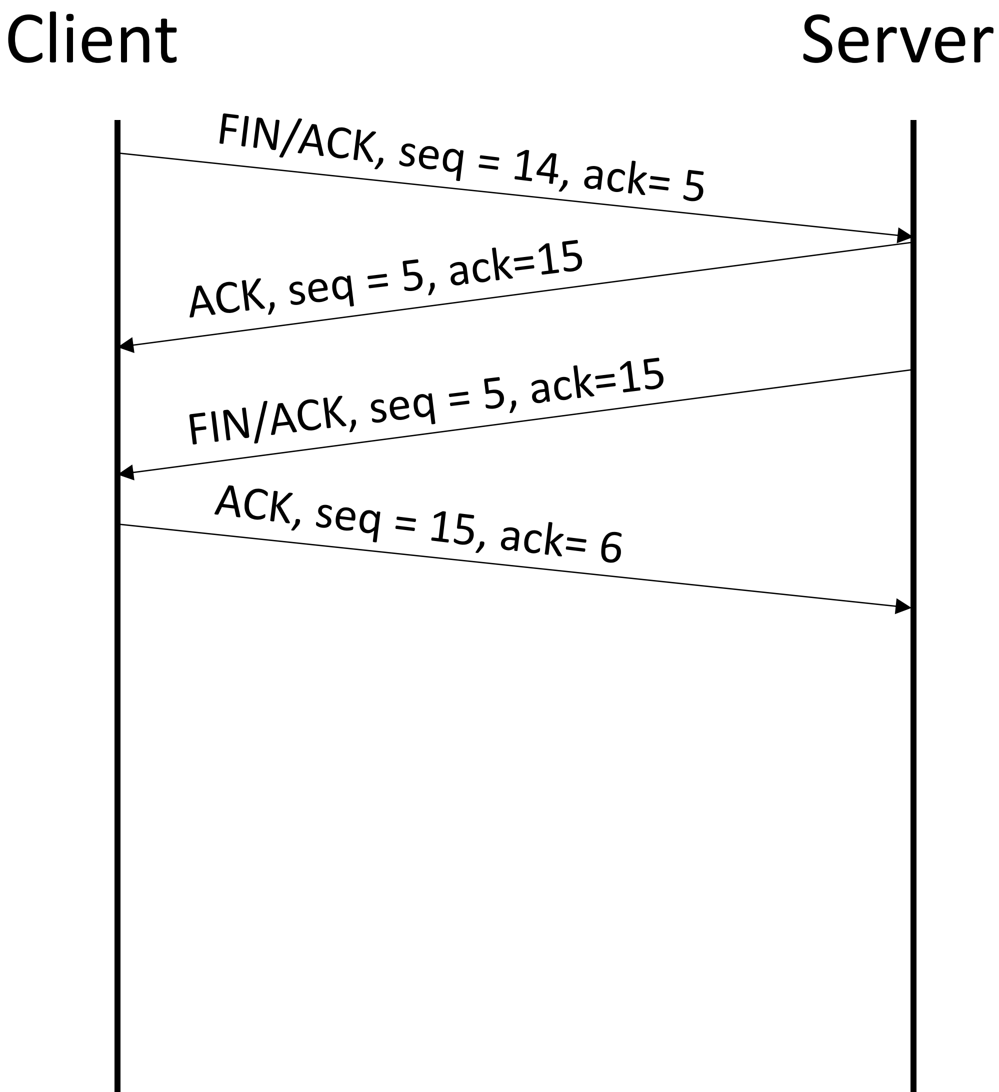
8) Server receives FIN packet and responds with ACK

9) Client receives ACK, says nothing

10) Server sends FIN/ACK

11) Client receives FIN/ACK and sends ACK

Connection Tear Down



7) Client sends packet to server with FIN/ACK flags set

8) Server receives FIN packet and responds with ACK

9) Client receives ACK, says nothing

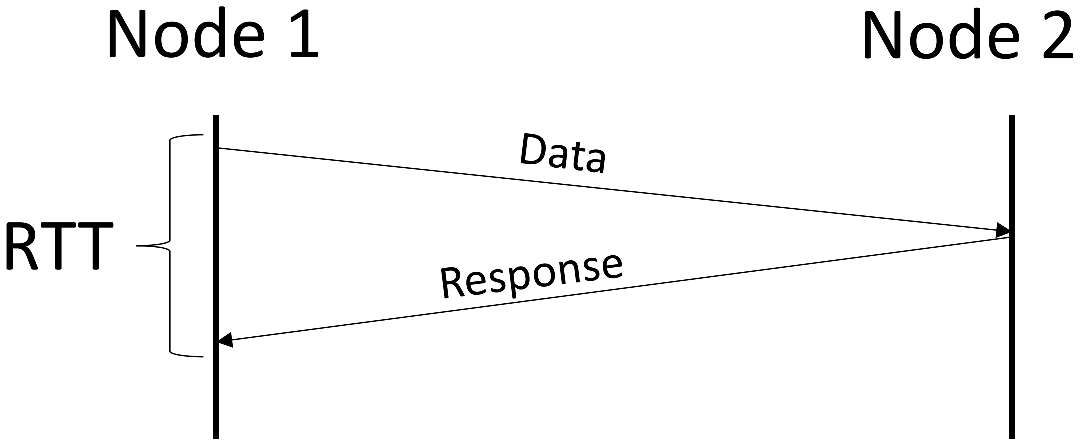
10) Server sends FIN/ACK

11) Client receives FIN/ACK and sends ACK

12) Server receives ACK and closes connection

13) When timer expires, client also transitions to closed

Congestion Control



- Expected time to receive a packet = RTT (determined initially by 3-way handshake)
- Window size controls sending rate
- Additive Increase, Multiplicative Decrease (AIMD)

Fast Retransmit

- Three duplicate acks = something is missing but congestion isn’t that bad because some packets are getting through
- Timeout = no packets are getting through, bad congestion

Summary

- TCP connection is started with 3-way handshake
- TCP connection is ended with 4-way handshake
- TCP congestion control uses AIMD to determine sending rate
- Congestion can be detected by three duplicate acks OR timeout