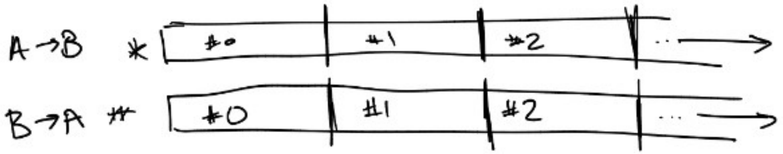
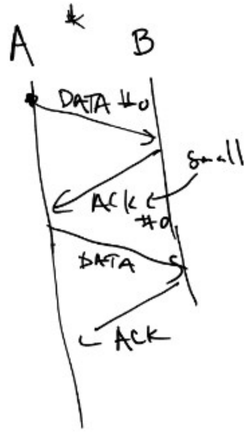
Lecture 23: TCP, Sockets

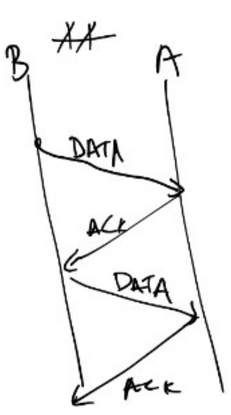
- TCP
  - Piggybacking, Windowing, timeouts

- Sockets
  - API for interacting with network

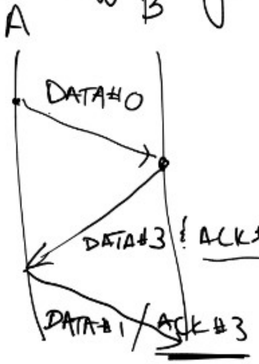
TCP (con't)





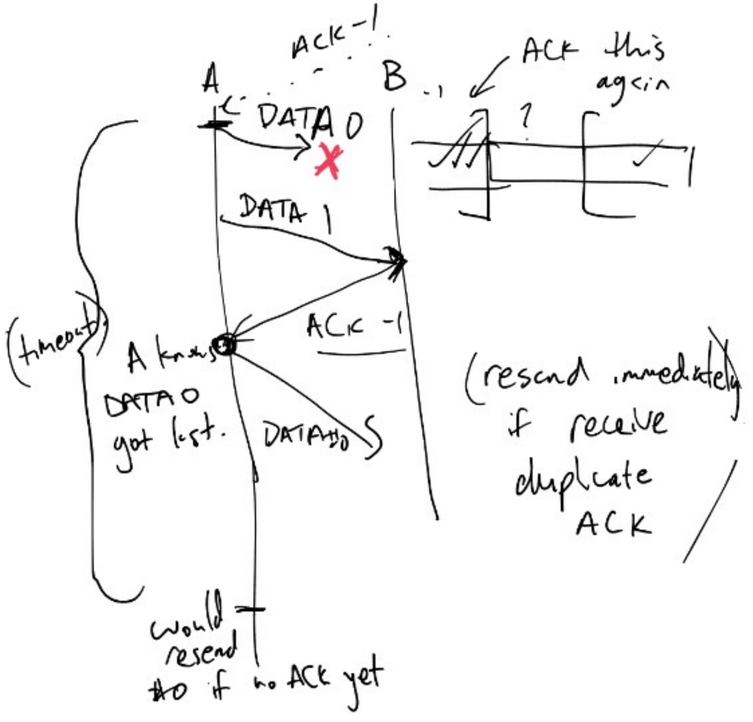


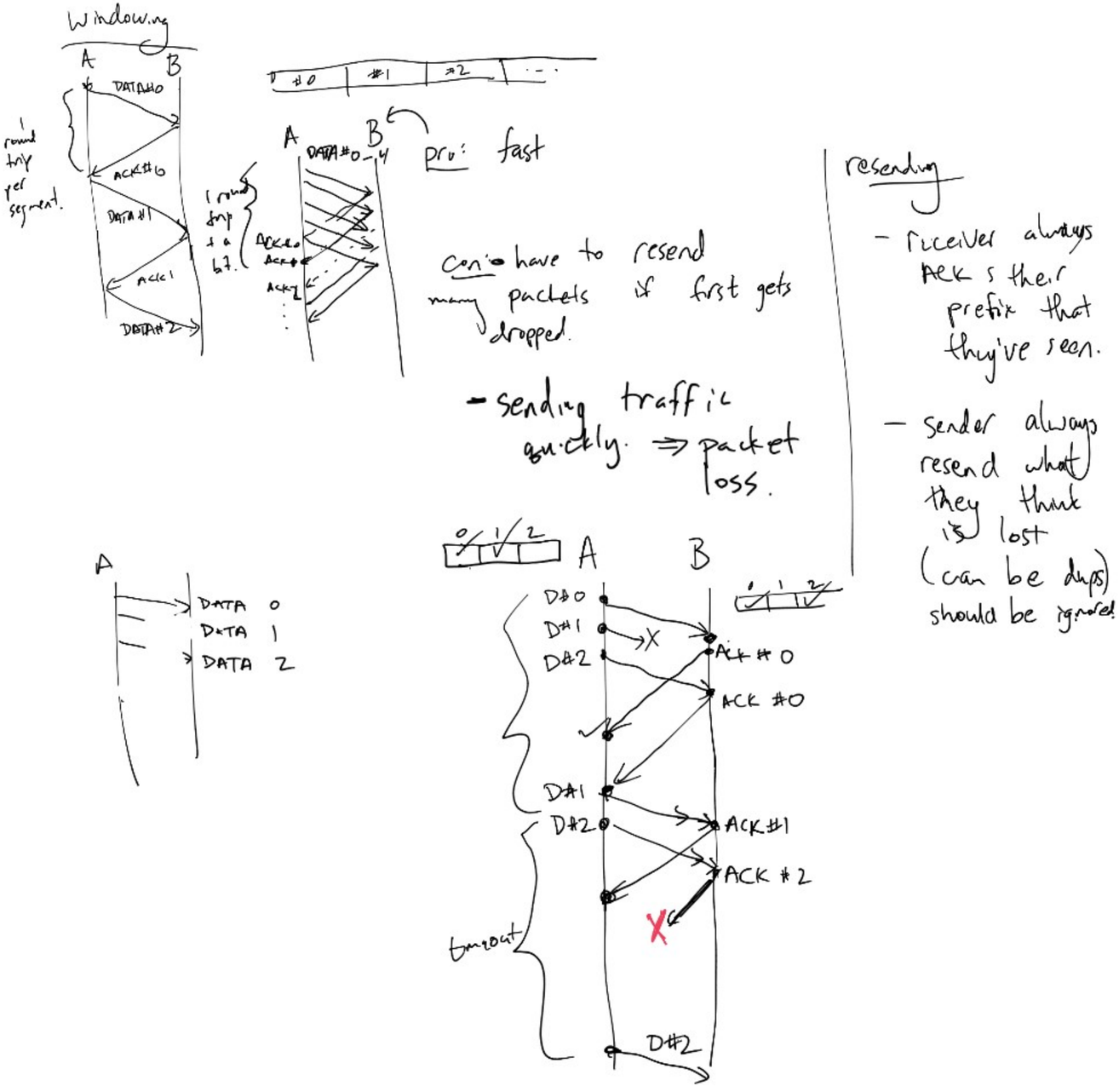
=> combine (piggybacking)



ack mean "I've received all packets from 0...n"

↳ A has received 0...3

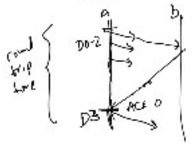



Windowing
A
B
DATA#0
ACK#0
DATA#1
ACK1
DATA#2
1 round trip per segment.
1 round trip + a bit.
DATA#0-4
ACK0
ACK1
ACK2
pro: fast
con: have to resend many packets if first gets dropped.
- sending traffic quickly. ⇒ packet loss.
resending
- receiver always ACKs their prefix that they've seen.
- sender always resend what they think is lost (can be dups) should be ignored
DATA 0
DATA 1
DATA 2
D#0
D#1
D#2
ACK #0
ACK #1
ACK #2
timeout
D#2

Managing traffic (congestion control)

TCP has "window size": # of unacked packets that are in transit

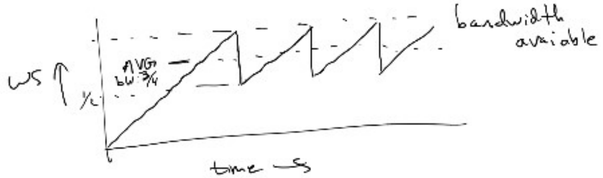
window size: 3

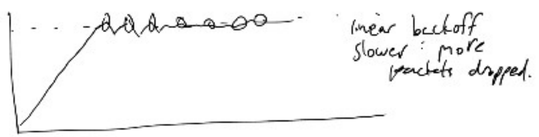


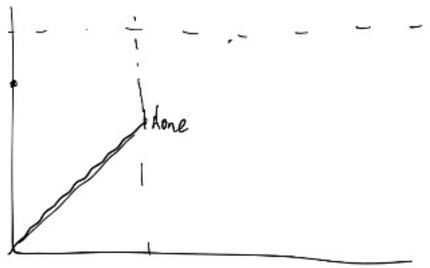
want window size to be exactly the # of packets you can send in 1 round trip time (measure of avail bandwidth).

Bandwidth: data rate
Latency: how long it takes to send 1 packet.

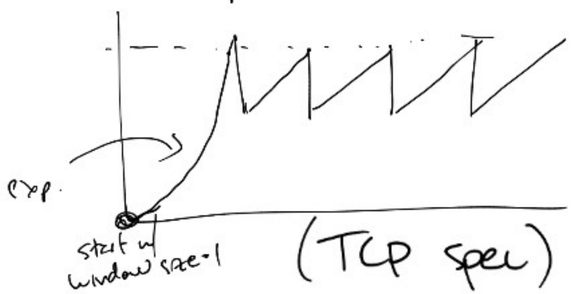
Can adjust window size
- increase when I successfully transmit data (increase a little: add 1 to WS.) — Linear increase
- decrease when data dropped (timeout / dup ACK) (decrease a lot: halve window size) — Exponential back off



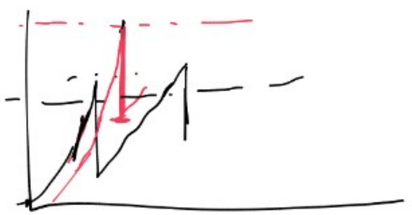




SLOW start (worst name ever)
: exponential increase until 1st drop



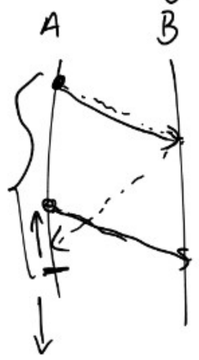
(TCP spec)



Timeouts:

Want to time out just after you would have received an ACK (RTT)

- timeout too long: spend time waiting
- too short: premature resends.



estimate RTT: every time you receive ACK, measure time since you sent DATA, give an estimate of RTT.

---

update your estimate of RTT,

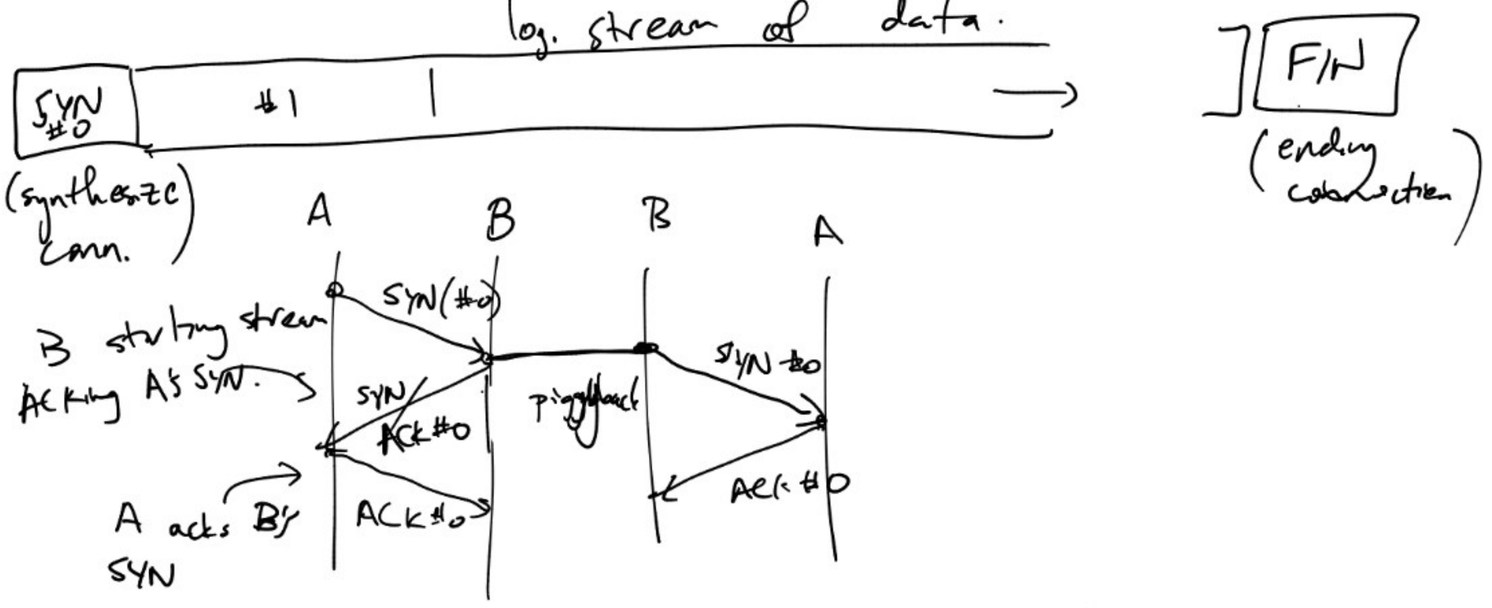
new est = α · old estimate + $(1-\alpha)$ · observed RTT

$0 \leq \alpha \leq 1$.

$\alpha = 0$: no history

$\alpha = 1$: ignoring observations.

Establishing / ending connections



"three-way handshake: SYN, SYN/ACK, ACK

Sockets & Multiplexing

- TCP communication endpoints identified by
  IP address + port #
  (identifies service)
  e.g. port 80: web traffic (HTTP)

- Multiple clients connect to port 80 on server.

- Client connects to port 80 from port (say 10,000)

- Server choose a high-numbered port arbitrarily, send SYN/ACK from high # port (e.g. 37373) back to client port 10,000

- Subsequently, all traffic is on port 37373 of server ↔ 10,000 of client

server socket

client sockets

server socket listens for incoming connections, creates client sockets.