CS1680 Spring 2026

April 21 Guest Lecture

"Pre-Reqs": TCP Project

- Implemented TCP sliding window
  - How big should the window be? Currently, based on flow control
- Socket API
  - in-order delivery
- Retransmitting lost packets

TCP and Sending Rate

What controls what our TCP *thoughput* is going to be?

- Break down this question: what input prevents our TCP implementation from sending more packets?
- How often does our sliding window advance?
  - One window per RTT

TCP Sending Rate = $\frac{CWND}{RTT}$

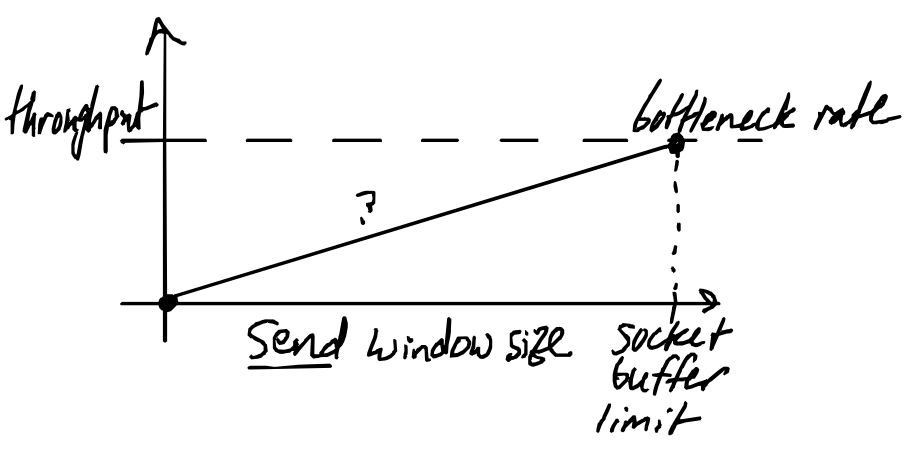
Currently, in your project implementations, you probably set the window size using *flow control*. The purpose of flow control is to make sure the receiver has enough memory available in its socket buffer to store the stuff we're sending it. Otherwise, we would waste a bunch of work because we wouldn't have a place to store the data once it arrived.

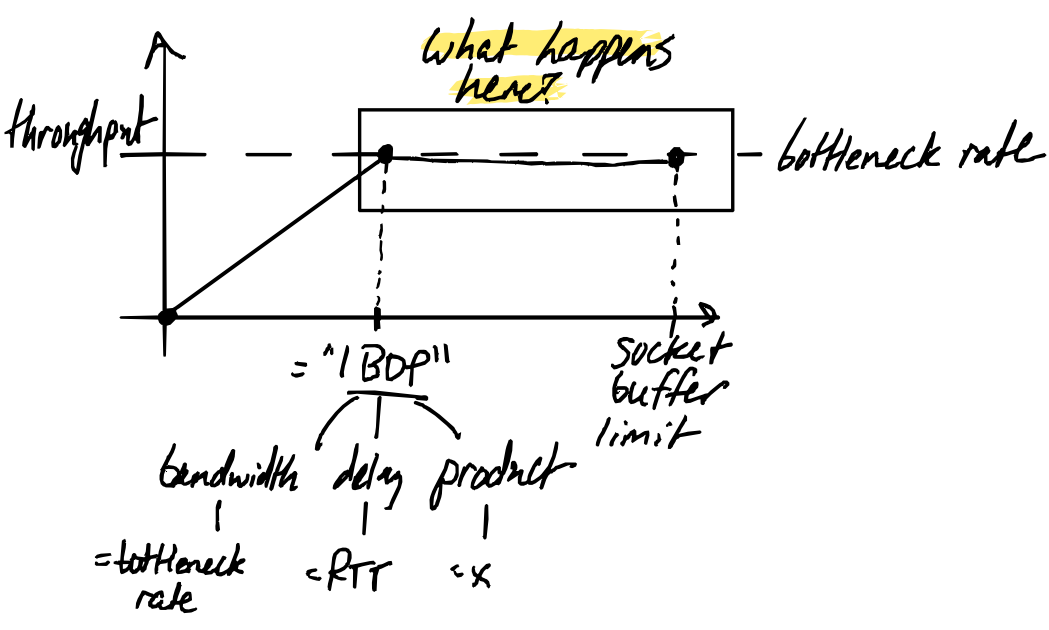
How big could we make our socket buffer (if we wanted)? How fast could we send?

- Modern machines have multiple GByte of memory - let's say our TCP receiver allocates a socket buffer of size 1 GByte.
- What's a normal Internet RTT?
  - ping Berkeley - ~80ms to Berkeley
- (window = 1 GByte) / (RTT = 80ms) = 100Gbit/s
  - Raise your hand if you have a 100Gbit/s Internet connection to Berkeley!
- So, our socket buffer + flow control isn't really determining how fast we can send data. Instead, what's limiting us?
  - The links in between (which other people might also want to use...)

How can we figure out how fast to send?

- (Other ideas?)
- Just try various sending rates and see. Set a "congestion window" (CWND)
  - See...what?

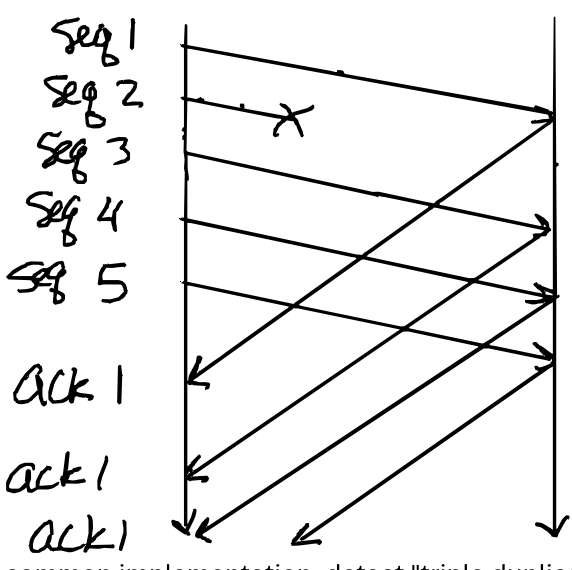




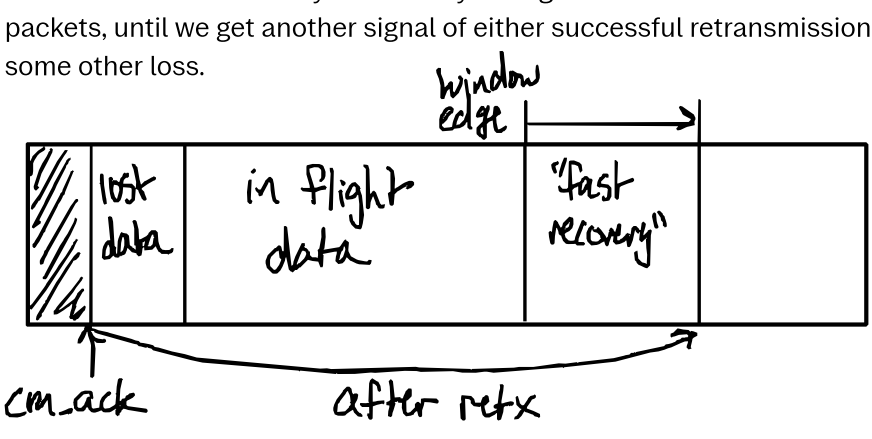
- Remember - we don't know what the bottleneck rate is. That's up to us to discover.
- Idea: probe from progressively smaller rates to larger ones until a loss happens.
- RTO-based loss not ideal for this - we'll have to wait a long time!

Detecting Loss

- What signals for packet loss, other than RTO timeout, might we be able to observe?



- common implementation: detect "triple duplicate ACK" as a signal of packet loss
- This still leaves us with a problem: While we are retransmitting that packet, our window still isn't sliding!
- Fix this with "fast recovery": artificially enlarge the window to send more packets, until we get another signal of either successful retransmission or some other loss.



When we've detect a loss and retransmitted the corresponding packet, what should we do with our sending rate?

Idea 1: reset to initial value.

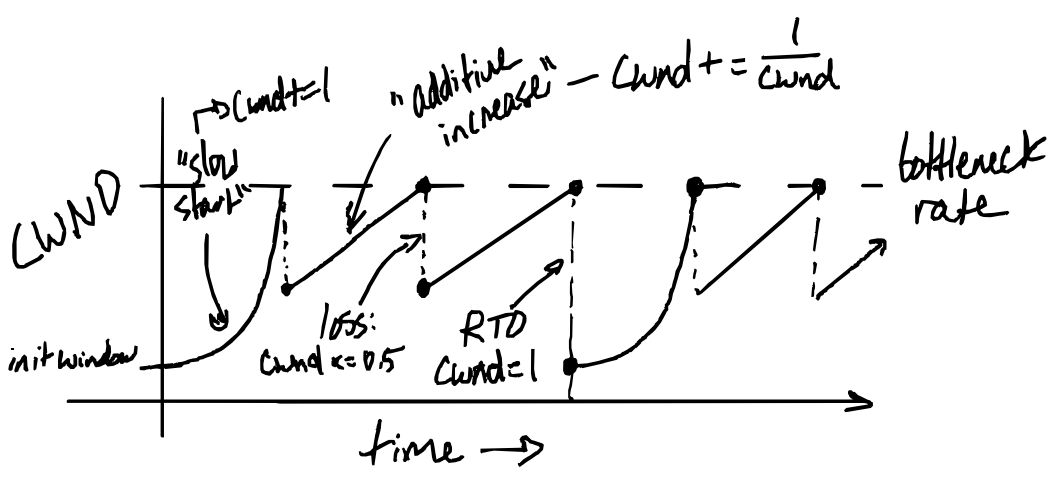
Idea 2: linearly decrease (-= 1)

Idea 3: multiplicatively decrease (/= 2)

- Turns out (Chiu-Jain '89) that "additive increase, multiplicative decrease" converges to fair bandwidth shares among flows
- Why this is the case is covered in 2680
- Called "AI-MD" for additive increase, multiplicative decrease

TCP "Sawtooth"

- Increase quickly during "slow start"
- Increase conservatively during "additive increase"
- Decrease multiplicatively on isolated packet loss
- Reset state on RTO



Delay-Based Congestion Control

What other signals than loss (and lack of loss) could we use to set our CWND?

The RTT:

- if cwnd > BDP, RTT will start increasing due to queueing delay
- Vegas: If (RTT - min(RTT) > 3*MSS) cwnd -- else cwnd ++

The ack-rate:

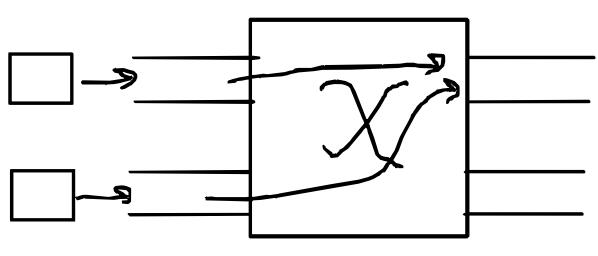
- ack-rate <= bottleneck rate, but send-rate can be higher
- BBR: send-rate *= 1.25; sleep(1 RTT); send-rate = max(ack-rate) * C
- C: how much queueing delay we want to have
- Kleinrock and queuing theory: 0 queueing delay is unachievable

These algorithms can more efficiently find the 0 queueing delay point, but it's harder for them to achieve bandwidth fairness.

In-Network AQM + Fairness

Another approach is to look at this problem from the routers themselves.

Packets arrive, but multiple packets for the same output link might show up at the same time (especially if they're trying to maximize their sending rates)



Ways of managing this:

- Input queueing: Head-of-line blocking
- Output queueing: Hard to implement in hardware
- Virtual output queueing to avoid head-of-line blocking

How do we manage the queue?

- easiest: FIFO (first in, first out)
- harder: group packets by class (5-tuple) and dequeue from classes fairly
  - DRR (Deficit Round Robin): discussed in 2680

Wrap-up

- Very hard to figure out what rate to send at (congestion control algorithms)
- Also very hard to manage this in-network
- Where to go for more?
  - Socket API and systems performance: **CS1675 (Fall 2026)**
  - Bandwidth and network management: **CS2680**

CS1680 Spring 2026

April 21 Guest Lecture (by Akshay)

"Pre-Reqs": TCP Project

- Implemented TCP sliding window
  - How big should the window be? Currently, based on flow control
- Socket API
  - in-order delivery

1675

- Retransmitting lost packets

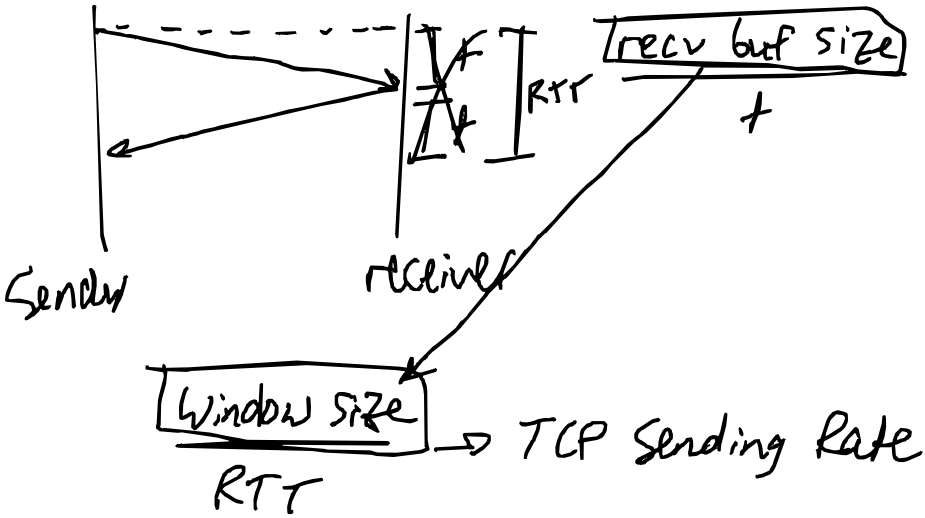
TCP and Sending Rate

bytes / second

What controls what our TCP *thoughput* is going to be?

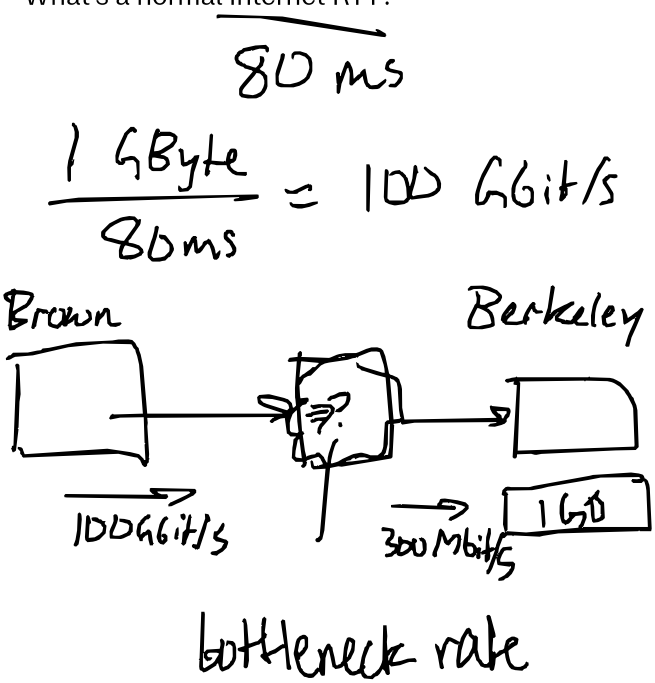
number of hosts

receive buffer size — bytes

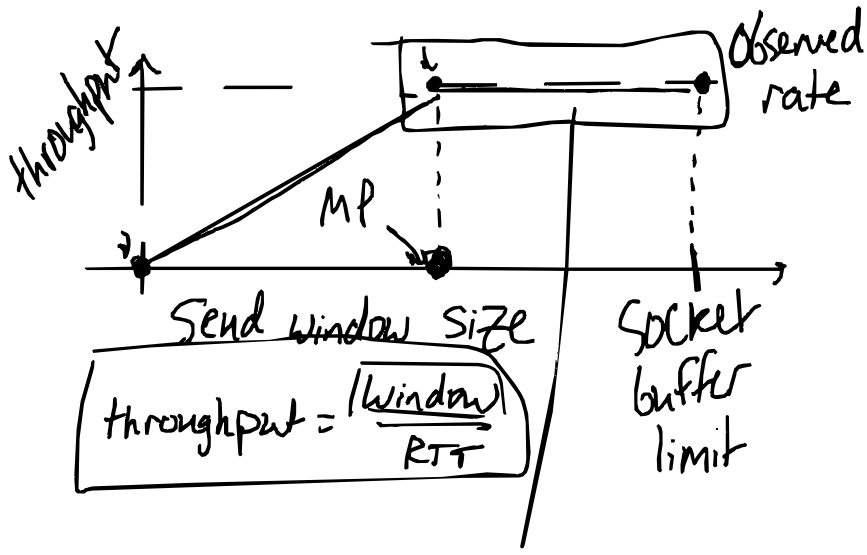


How big could we make our socket buffer (if we wanted)? How fast could we send?

- Modern machines have multiple GByte of memory - let's say our TCP receiver allocates a socket buffer of size 1 GByte.
- What's a normal Internet RTT?



How can we figure out how fast to send?



1. What happens here?
   - ↳ ISP does something?
   - ↳ store extra packets or delete (queue)
2. where is MP?
   - bottleneck rate

Window = ~~throughput~~ × RTT

BDP bandwidth delay product

↳ how to discover the BDP value?

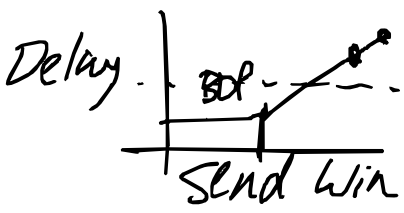
↳ increase window size and see what happens

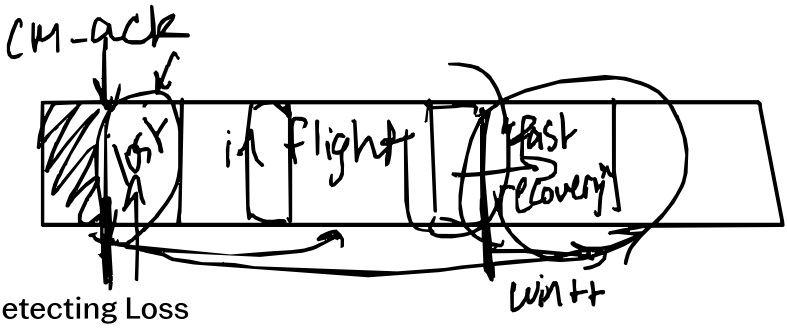
↳ packet loss

window size > BDP + queue size

↳ drop packets

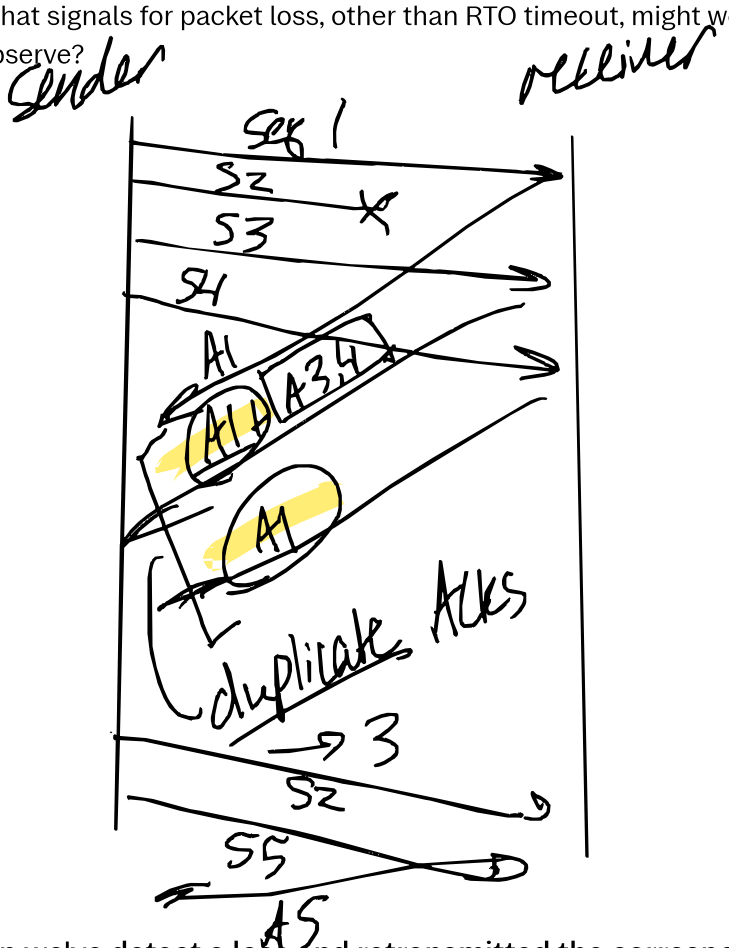
RTT = propagation delay + queueing delay





Detecting Loss

- What signals for packet loss, other than RTO timeout, might we be able to observe?



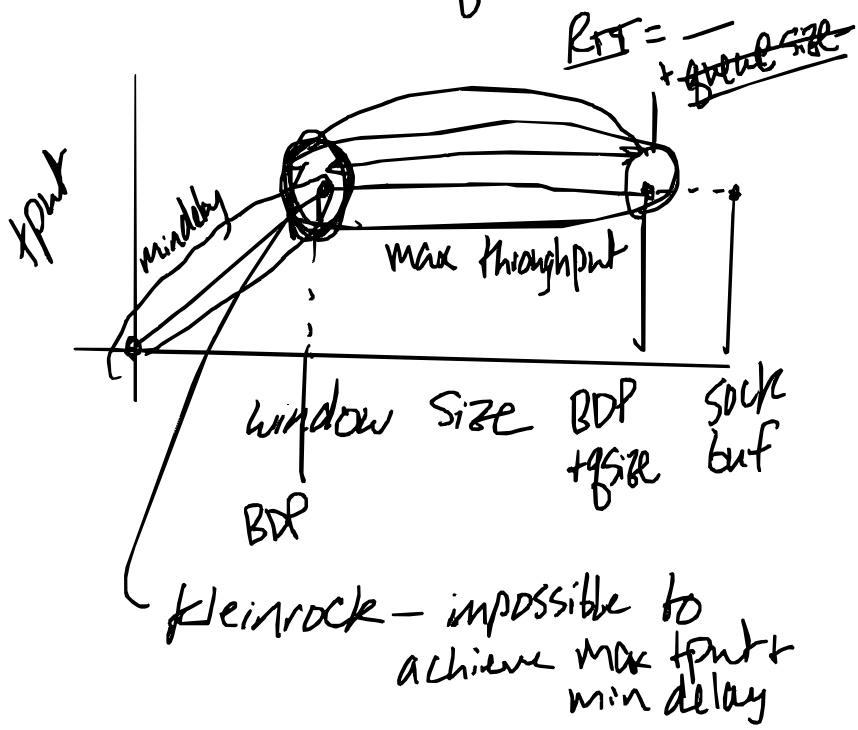
When we've detect a loss and retransmitted the corresponding packet, what should we do with our sending rate?

Idea 1: reset to initial value.

Idea 2: linearly decrease (-= 1)

Idea 3: multiplicatively decrease (/= 2)

Little's law
queue delay = size / dep. rate



TCP "Sawtooth"

- Increase quickly during "slow start"
- Increase conservatively during "additive increase"
- Decrease multiplicatively on isolated packet loss
- Reset state on RTO

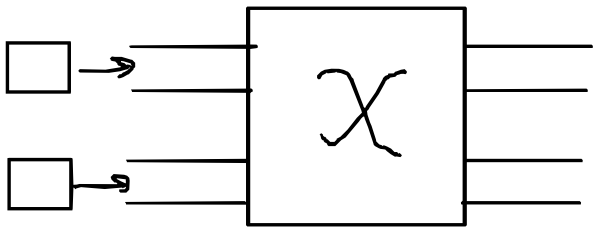
Delay-Based Congestion Control

What other signals than loss (and lack of loss) could we use to set our CWND?

In-Network AQM + Fairness

Another approach is to look at this problem from the routers themselves.

Packets arrive, but multiple packets for the same output link might show up at the same time (especially if they're trying to maximize their sending rates)



Ways of managing this?

How do we manage the queue?

Wrap-up

- Very hard to figure out what rate to send at (congestion control algorithms)
- Also very hard to manage this in-network
- Where to go for more?
  - Socket API and systems performance: **CS1675 (Fall 2026)**
  - Bandwidth and network management: **CS2680**