TCP Congestion Control

CSE 118: Computer Networks

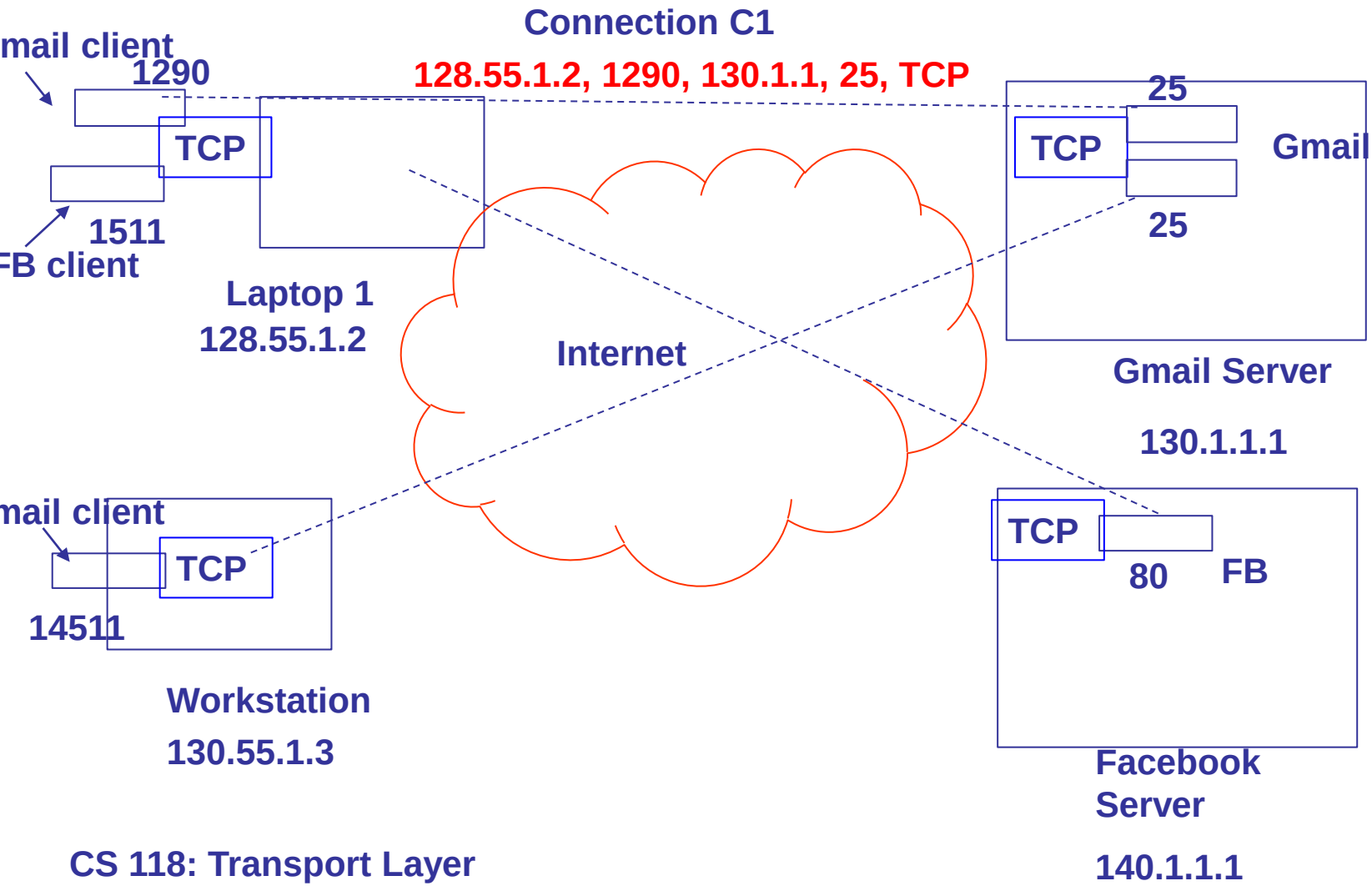
George Varghese

(some slides by Alex Snoeren)

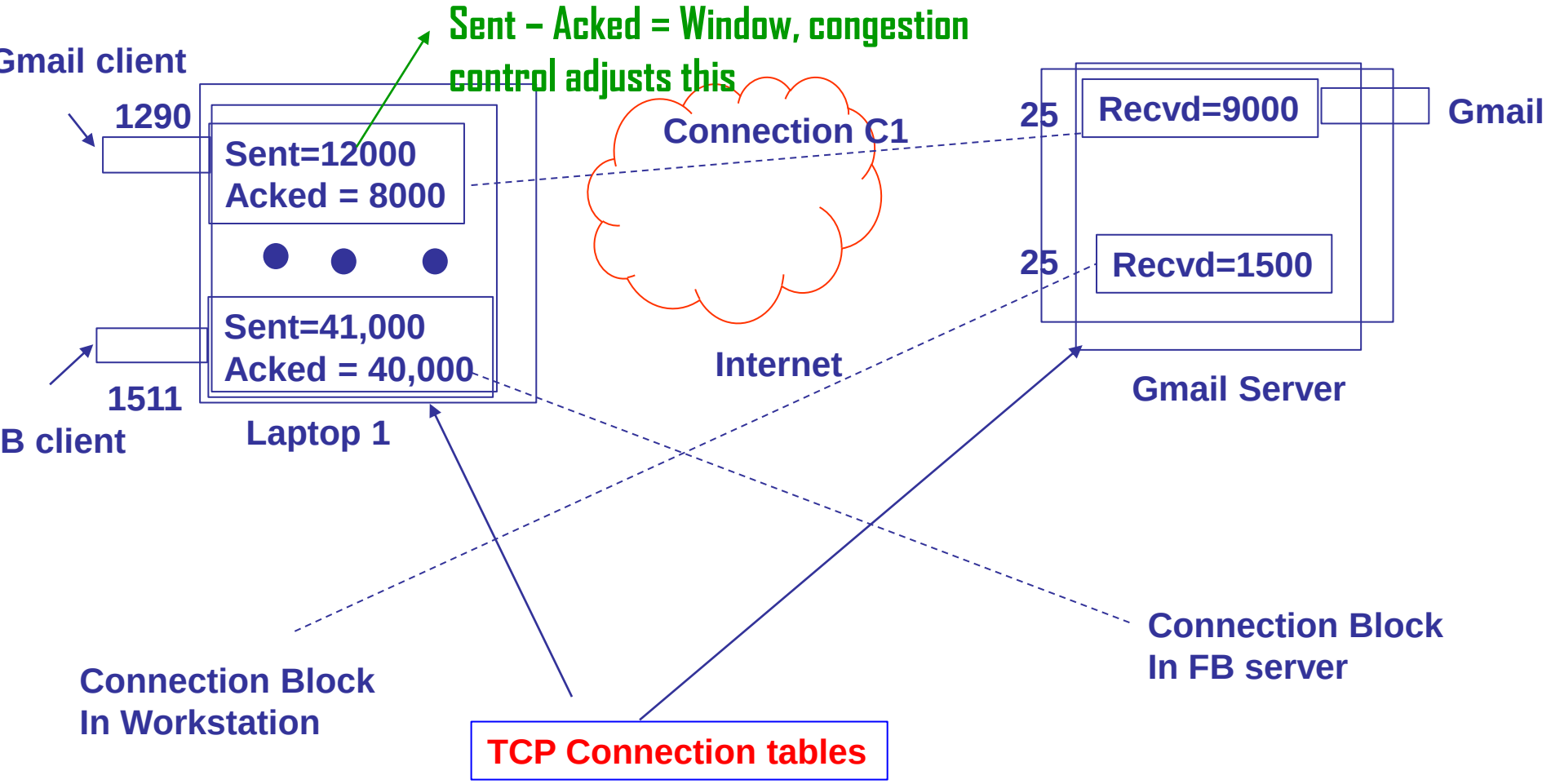
(including the latest updates from industry like Google's BBR and Microsoft's DCTCP)


ARE YOU ILL?
NO, JUST FEELING A BIT OFF.
11111111
11101111

Review


Connection C1
128.55.1.2, 1290, 130.1.1, 25, TCP
mail client
1290
TCP
1511
FB client
Laptop 1
128.55.1.2
Internet
25
TCP
Gmail
25
Gmail Server
130.1.1.1
mail client
TCP
14511
Workstation
130.55.1.3
TCP
80
FB
Facebook
Server
140.1.1.1

Connections between state variables in TCP tables



Flow vs Congestion Control

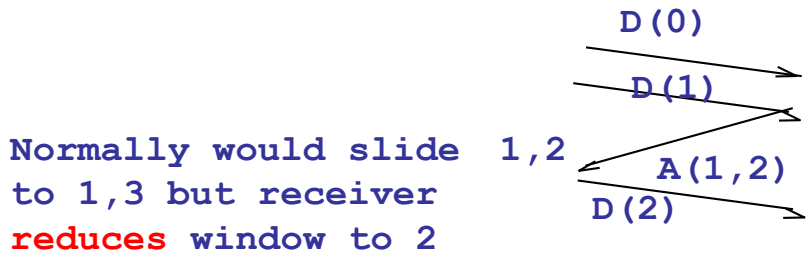
Flow Control: Matching speed of sender to receiver speed
Congestion Control: Matching speed of sender to network speed

He saved the Internet (1980s)

Van Jacobson, UCLA Affiliate Professor

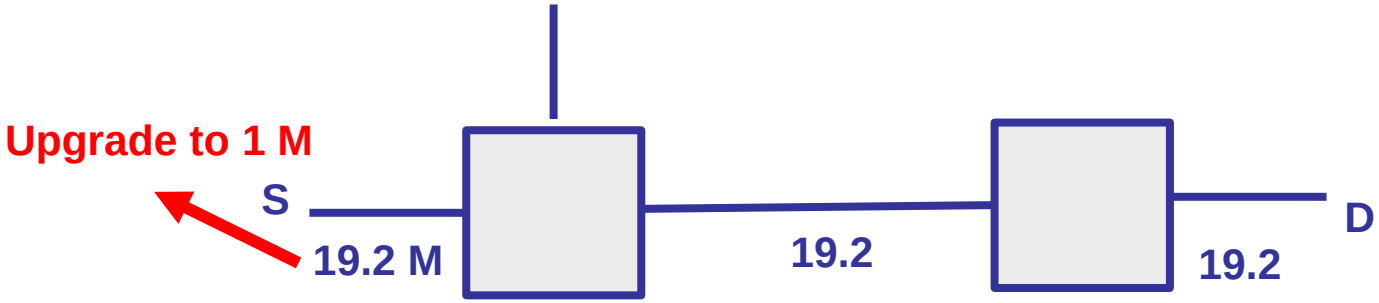
FLOW CONTROL

Windows provide static flow control. Can provide dynamic flow control if receiver acks indicate Lower and Upper Window Edge.



Need to avoid deadlock if window is reduced to 0 and then increase to c > 0. In OSI, receiver keeps sending c. In TCP, sender periodically probes an empty window.

WHY CONGESTION CONTROL IS NEEDED

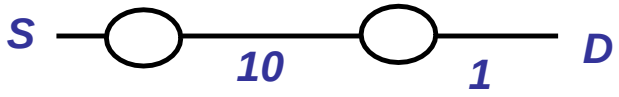


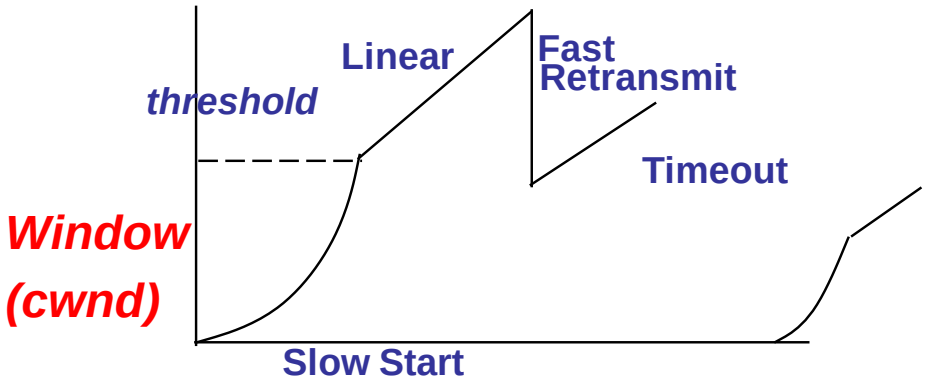
1) WHEN LINK FROM S TO FIRST ROUTER WAS UPGRADED FROM 19.2 KBPS TO 1 MBPS, THE TIME FOR A FILE TRANSFER WENT UP FROM A FEW SECONDS TO A FEW HOURS

2) THIS HAPPENED IN AN EXPERIMENT IN DEC IN THE 1980s. SHOWED THE NEED FOR CONGESTION CONTROL (DECBIT)

3) VERY SIMILAR EXPERIENCES IN INTERNET LED VAN JACOBSON TO PROPOSE TCP CONGESTION CONTOL. VAN IS AN AFFILIATE PROFESSOR AT UCLA!

TCP CONGESTION CONTROL SUMMARY

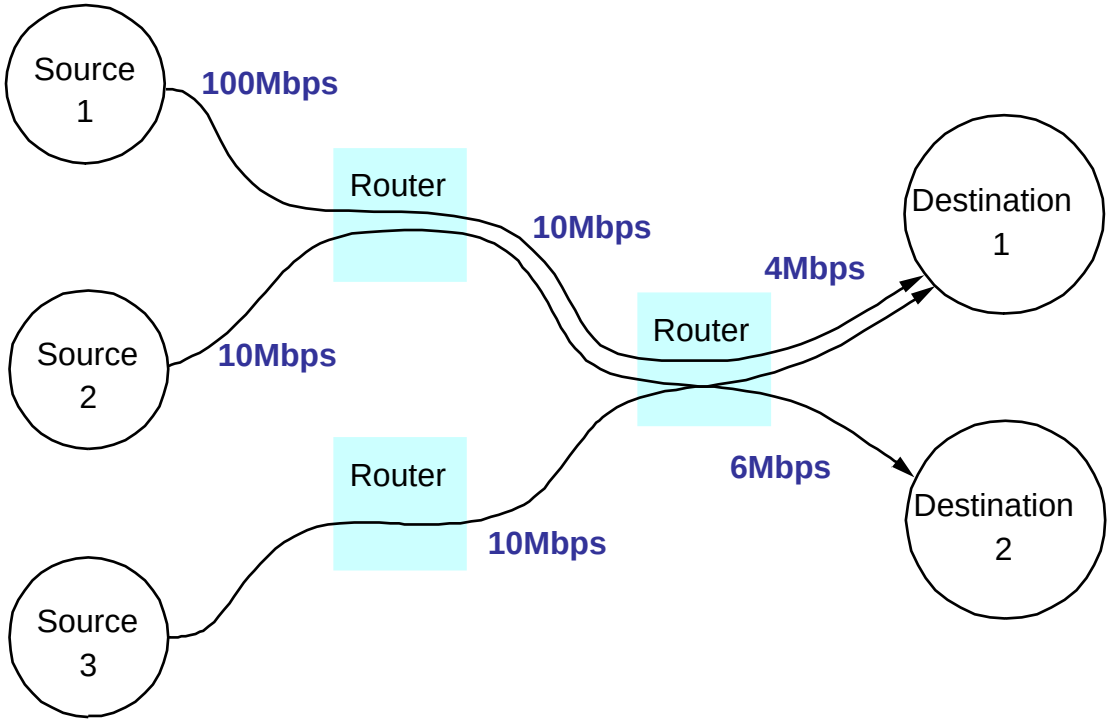




Congestion Control Overview

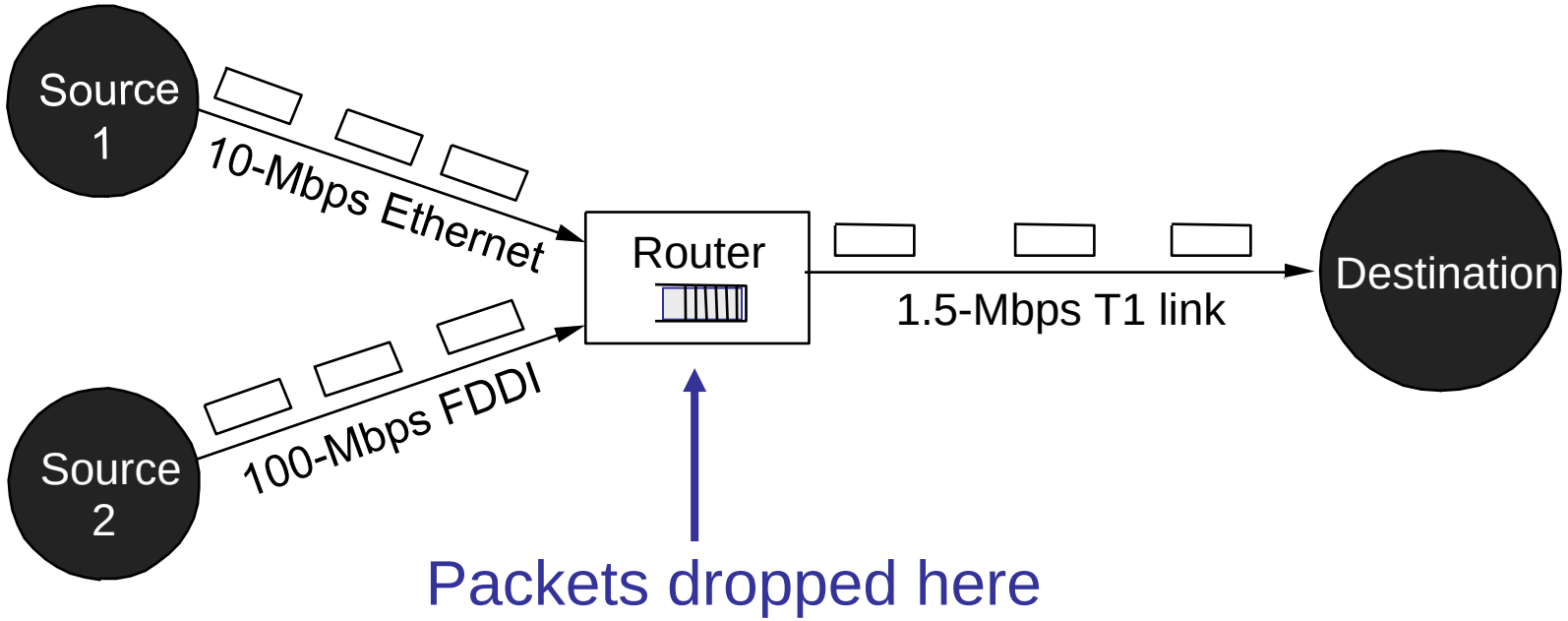
- How fast should a sending host transmit data?
  - Not too fast, not to slow, just right…
- Should not be faster than the sender's share
  - Bandwidth allocation
- Should not be faster than the network can process
  - Congestion control
- Congestion control & bandwidth allocation are separate ideas, but frequently combined

Fair Bandwidth Allocation



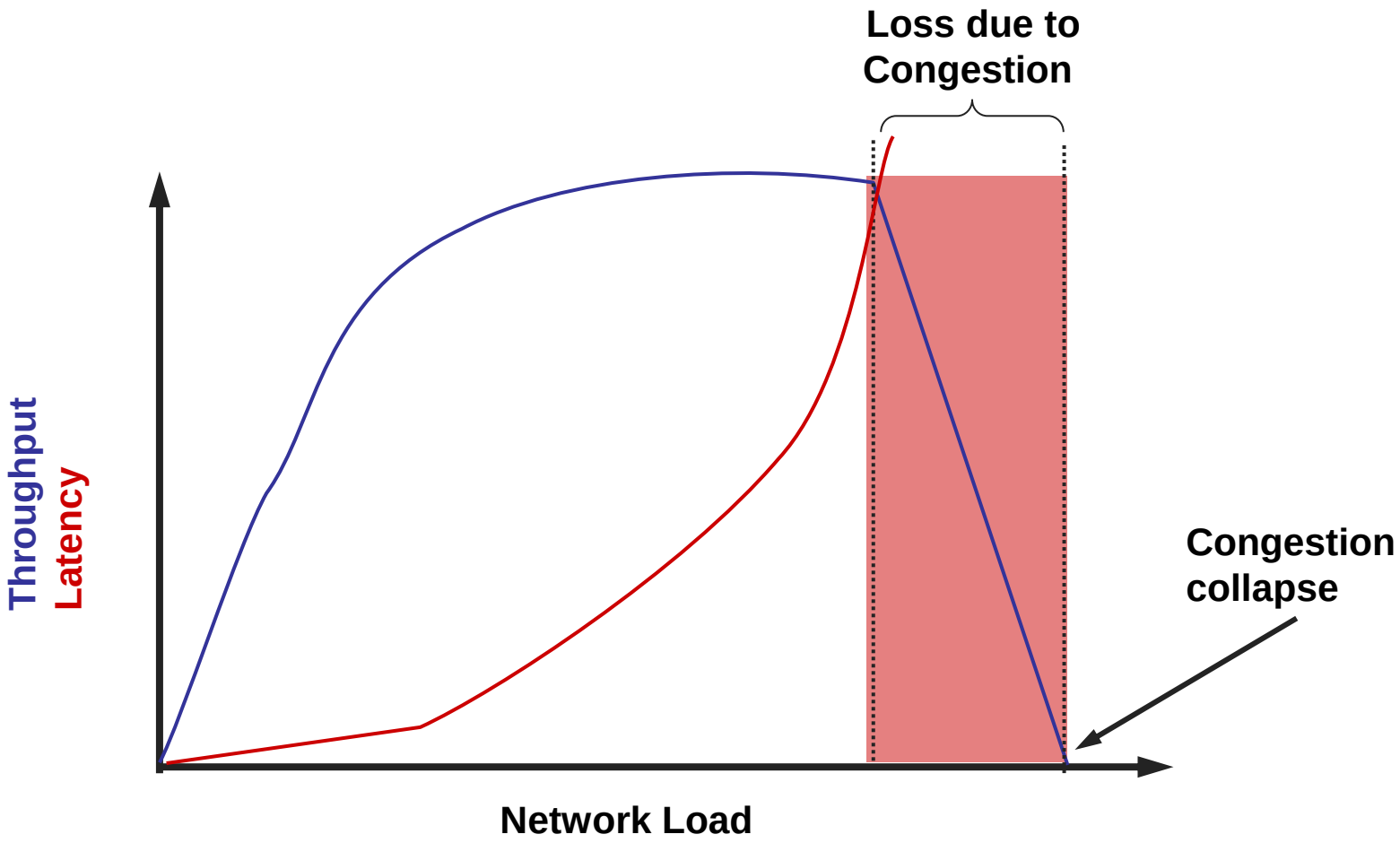
- How much bandwidth should each flow from a source to a destination receive when they compete for resources?
  - What is a “fair” allocation?

Congestion

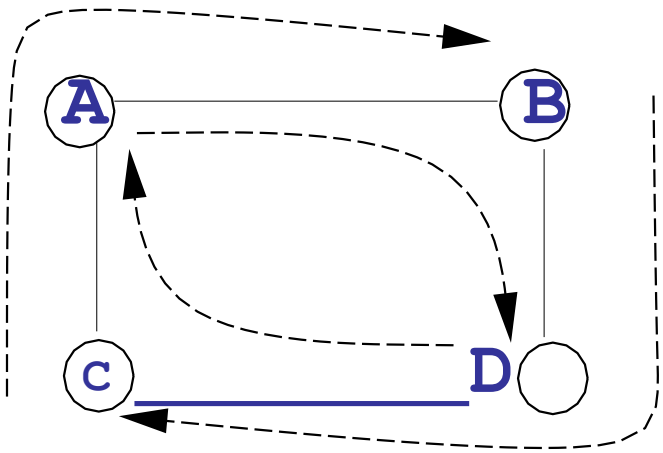


- Buffer intended to absorb bursts when input rate > output
- But if sending rate is persistently > drain rate, queue builds
- Dropped packets represent wasted work; goodput < throughput

Uncontrolled congestion?



WHY CONGESTION COLLAPSE CAN OCCUR



All packets go one hop and get dropped because of congestion
So no packets make progress. Goodput goes to 0,
Like gridlock in LA but more like livelock here.

Congestion Collapse

- Rough definition: “When an increase in network load produces a decrease in useful work”

- Why does it happen?
  - Sender sends faster than bottleneck link speed
  - Packets queue until dropped
  - In response to packets being dropped, sender retransmits
  - All hosts repeat in steady state…

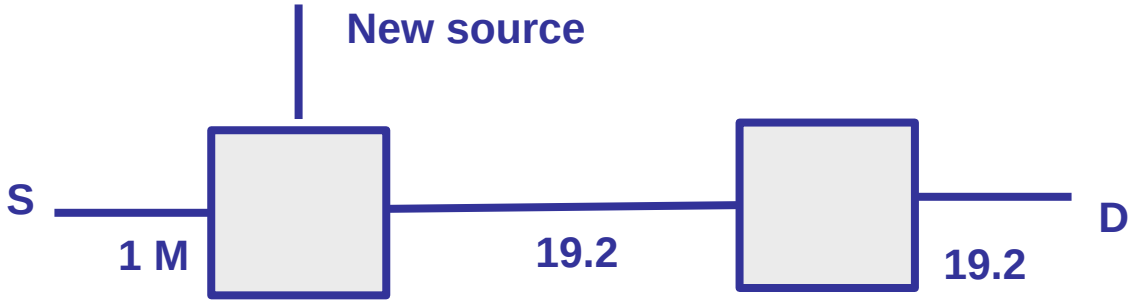
Mitigation Options

- Increase network resources
  - More buffers for queuing
  - Increase link speed
  - Pros/Cons of these approaches?
- Reduce network load (TCP strategy)
  - Send data more slowly
  - How much more slowly?
  - When to slow down?

Designing a Control Scheme

- Open loop
  - Explicitly reserve bandwidth in the network in advance
- Closed loop (TCP Strategy)
- Respond to feedback and adjust bandwidth allocation
- Network-based
  - Network implements and enforces bandwidth allocation
- Host-based (TCP Strategy)
  - Hosts are responsible for controlling their sending rate

FEEDBACK CONGESTION CONTROL



1) DETECT CONGESTION

2) FEEDBACK INFORMATION TO THE SOURCE

3) SOURCE ADJUSTS WINDOW: INCREASE POLICY
DECREASE POLICY

TWO INTERESTING CASES:

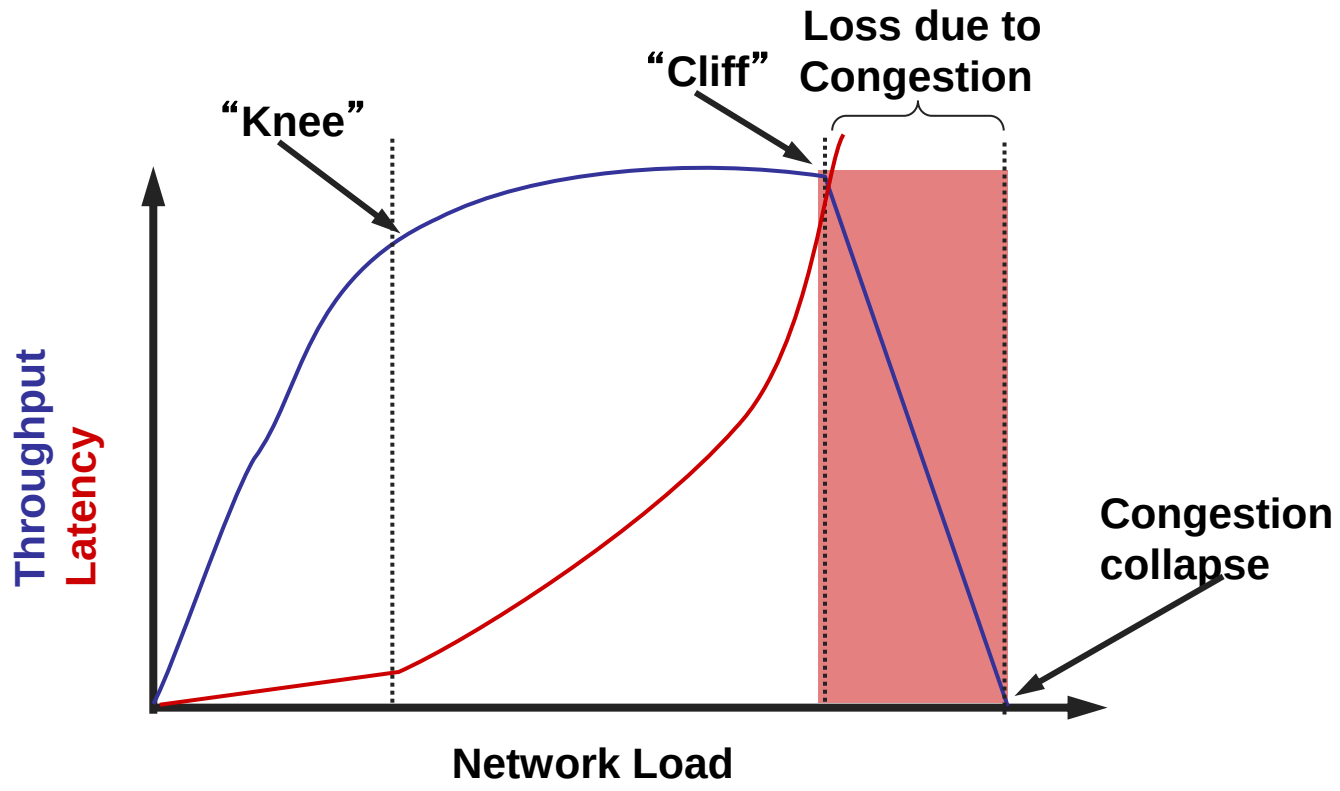
a) HOW A SOURCE REACHES STEADY STATE.

b) HOW A SOURCE REACTS TO A NEW SOURCE TO PROVIDE A FAIR ALLOCATION.

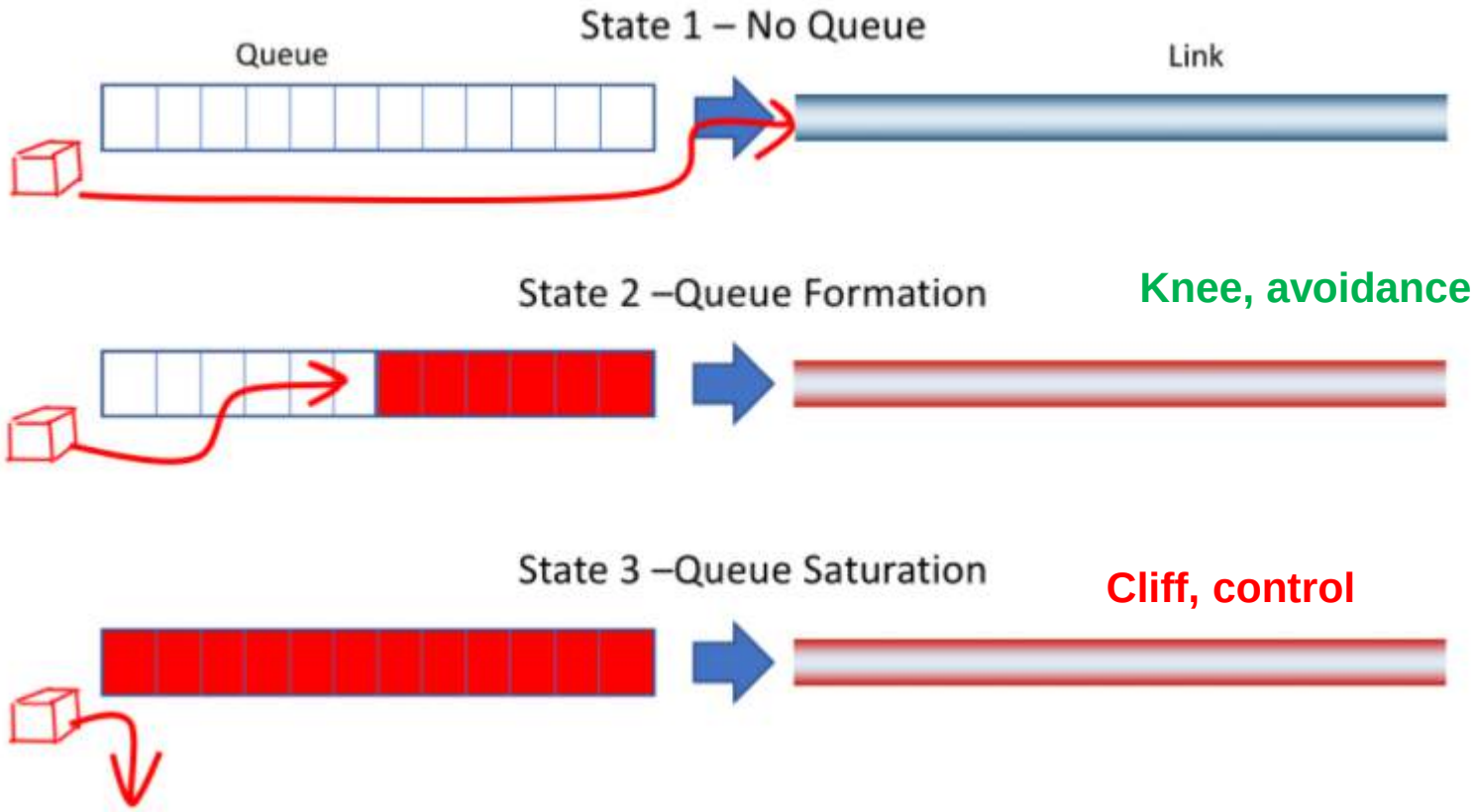
CONGESTION AVOIDANCE VERSUS CONGESTION CONTROL

Proactive *vs.* Reactive

- Congestion avoidance: try to stay to the left of the knee
- Congestion control: try to stay to the left of the cliff



Avoidance vs Control



From Huston's APNIC Blog, see later for URL

Detecting Congestion

- Explicit congestion signaling
  - Source Quench: ICMP message from router to sender
  - DECBit / Explicit Congestion Notification (ECN):
    - Router *marks* packet based on queue occupancy (i.e. indication that packet encountered congestion along the way)
    - Receiver tells sender if queues are getting too full
- Implicit congestion signaling
  - Packet loss
    - Assume congestion is primary source of packet loss
    - Lost packets indicate congestion
  - Packet delay
    - Round-trip time increases as packets queue
    - Packet inter-arrival time is a function of bottleneck link

Throttling Options

- Window-based (TCP)
  - Constrain number of outstanding packets allowed in network
  - Increase window to send faster; decrease to send slower
  - Pro: Cheap to implement, good failure properties
  - Con: Creates traffic bursts (requires bigger buffers)

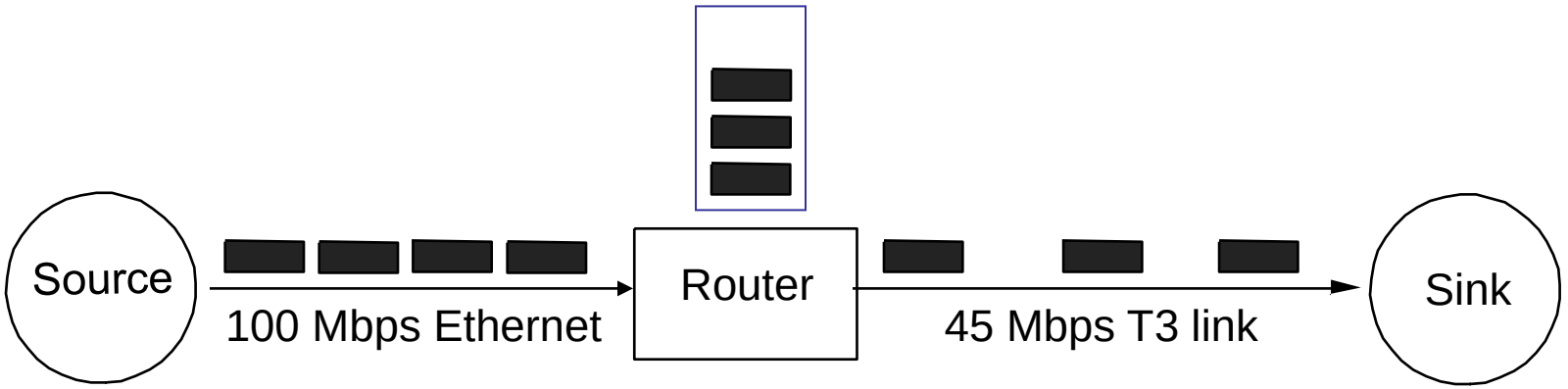
- Rate-based **(**many streaming media protocols, BBR**)**
  - Two parameters (period, packets)
  - Allow sending of x packets in period y
  - Pro: smooth traffic
  - Con: fine-grained per-connection timers, what if receiver fails?

Choosing a Send Rate

- Ideally: Keep equilibrium at “knee” of power curve
  - Find “knee” somehow
  - Keep number of packets “in flight” the same
  - Don’t send a new packet into the network until you know one has left (I.e. by receiving an ACK)
  - What if you guess wrong, or if bandwidth availability changes?

- Compromise: adaptive approximation
  - If congestion signaled, reduce sending rate by *x*
  - If data delivered successfully, increase sending rate by *y*
  - How to relate *x* and *y*? Most choices don’t converge…

TCP's Probing Approach



- Each source independently probes the network to determine how much bandwidth is available
  - Changes over time, since everyone does this
- Assume that packet loss implies congestion
  - Since errors are rare; also, requires no support from routers

Basic TCP Algorithm

- Window-based congestion control
  - Unified congestion control and flow control mechanism
  - *rwin*: advertised flow control window from receiver
  - *cwnd*: congestion control window
    - » Estimate of how much outstanding data network can deliver in a round-trip time
  - Sender can only send MIN(*rwin,cwnd*) at any time
- Idea: decrease cwnd when congestion is encountered; increase cwnd otherwise
  - Question: how much to adjust?

Congestion Avoidance

- Goal: Adapt to changes in available bandwidth
- Additive Increase, Multiplicative Decrease (AIMD)
  - Increase sending rate by a constant (e.g. MSS)
  - Decrease sending rate by a linear factor (e.g. divide by 2)
- Rough intuition for why multiplicative is good
  - One source reaches steady state window of 24
  - New source comes up and starts with window of 1
  - When both increase and detect congestion, first source cuts to 12 and other to 1. Eventually they equalize. Fairness!

TCP Bandwidth Probing

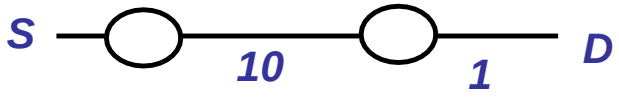
- TCP uses AIMD to adjust congestion window
  - Converges to fair share of bottleneck link
  - Increases modestly in good times
  - Cuts drastically in bad times

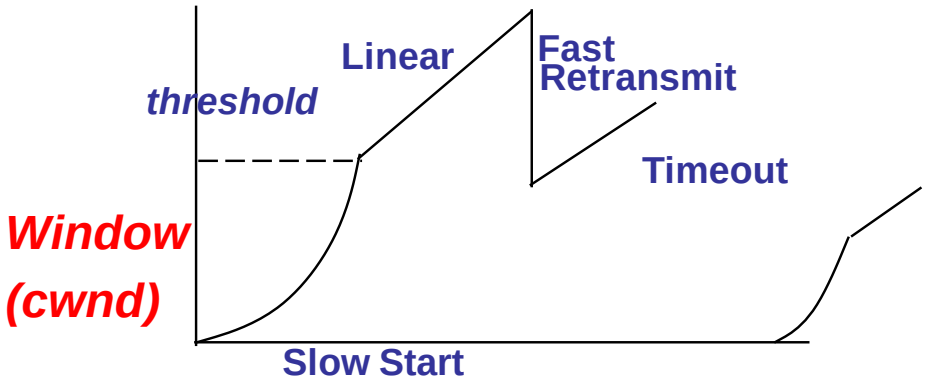
- But what rate should a TCP flow use initially?
  - Need some initial congestion window
  - We'd like to TCP to work on all manner of links
  - Need to span 6+ orders of magnitude, e.g., 10 K to 10 Gbps.
  - Starting too fast is catastrophic! So start with cwnd = 1

Slow Start

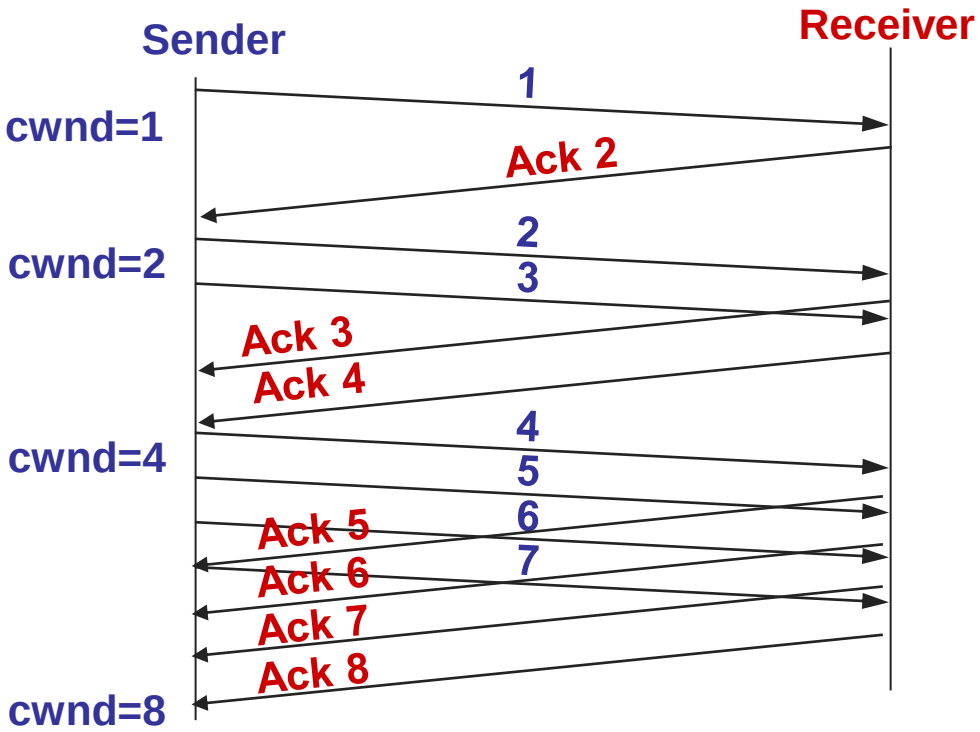
- Goal: quickly find the equilibrium sending rate
- Quickly increase sending rate until congestion detected
  - Remember last rate that worked and don't overshoot it
- TCP Reno Algorithm:
  - On new connection, or after timeout, set *cwnd*=1 MSS
  - For each segment acknowledged, increment *cwnd* by 1 MSS
  - If timeout then divide *cwnd* by 2, and set *ssthresh* = *cwnd*
  - If *cwnd* >= *ssthresh* then exit slow start
- Why called slow? Its exponential after all…

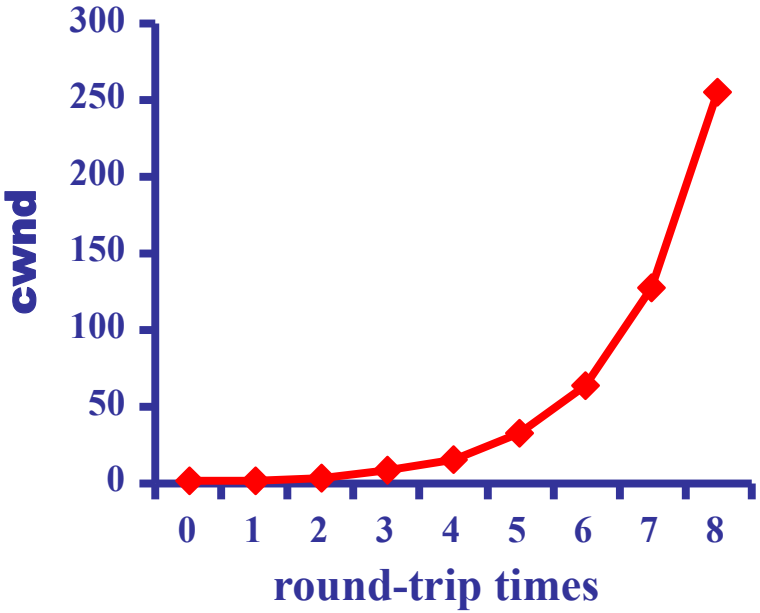
TCP CONGESTION CONTROL SUMMARY



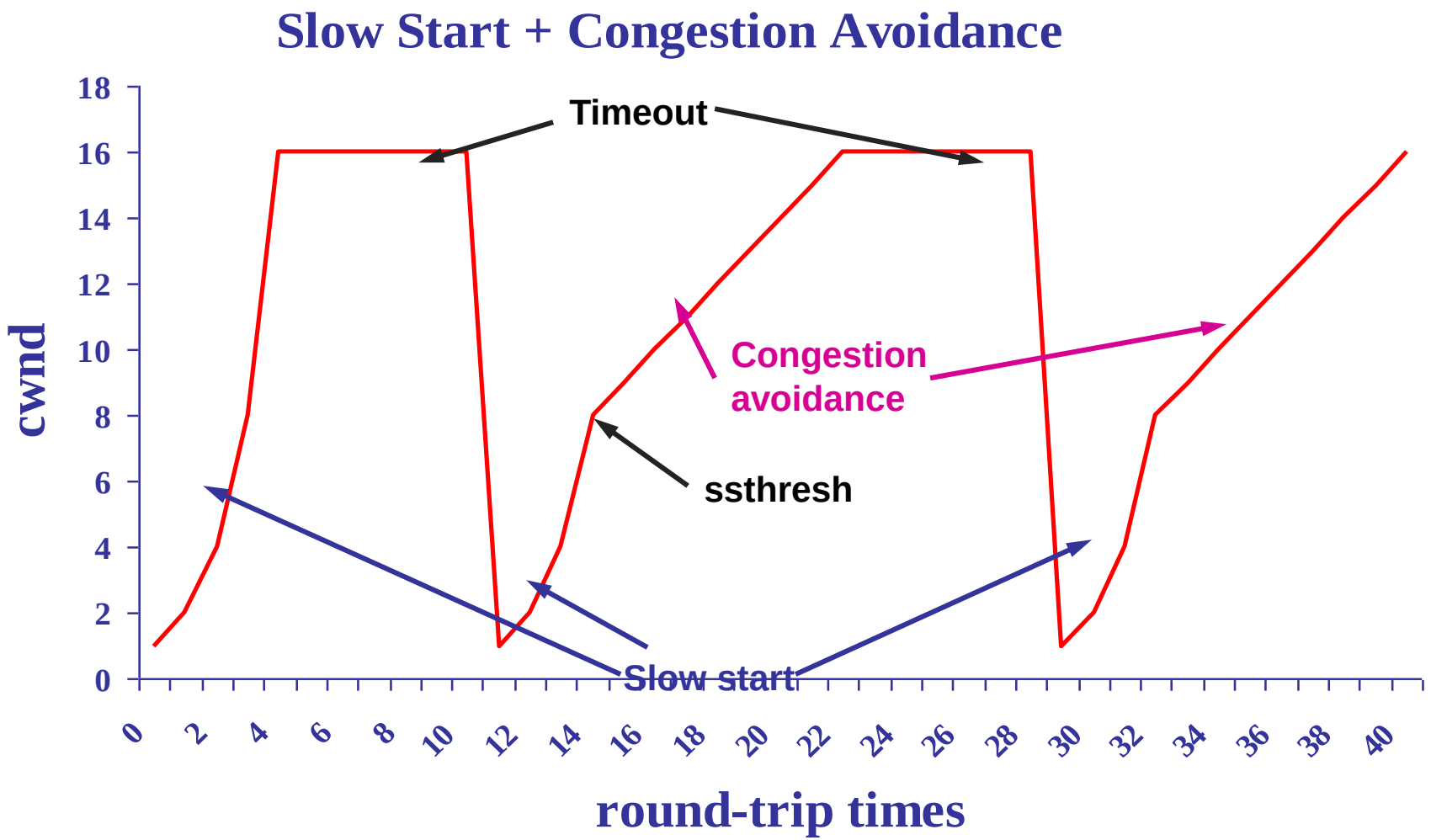


Slow Start Example





Basic Mechanisms

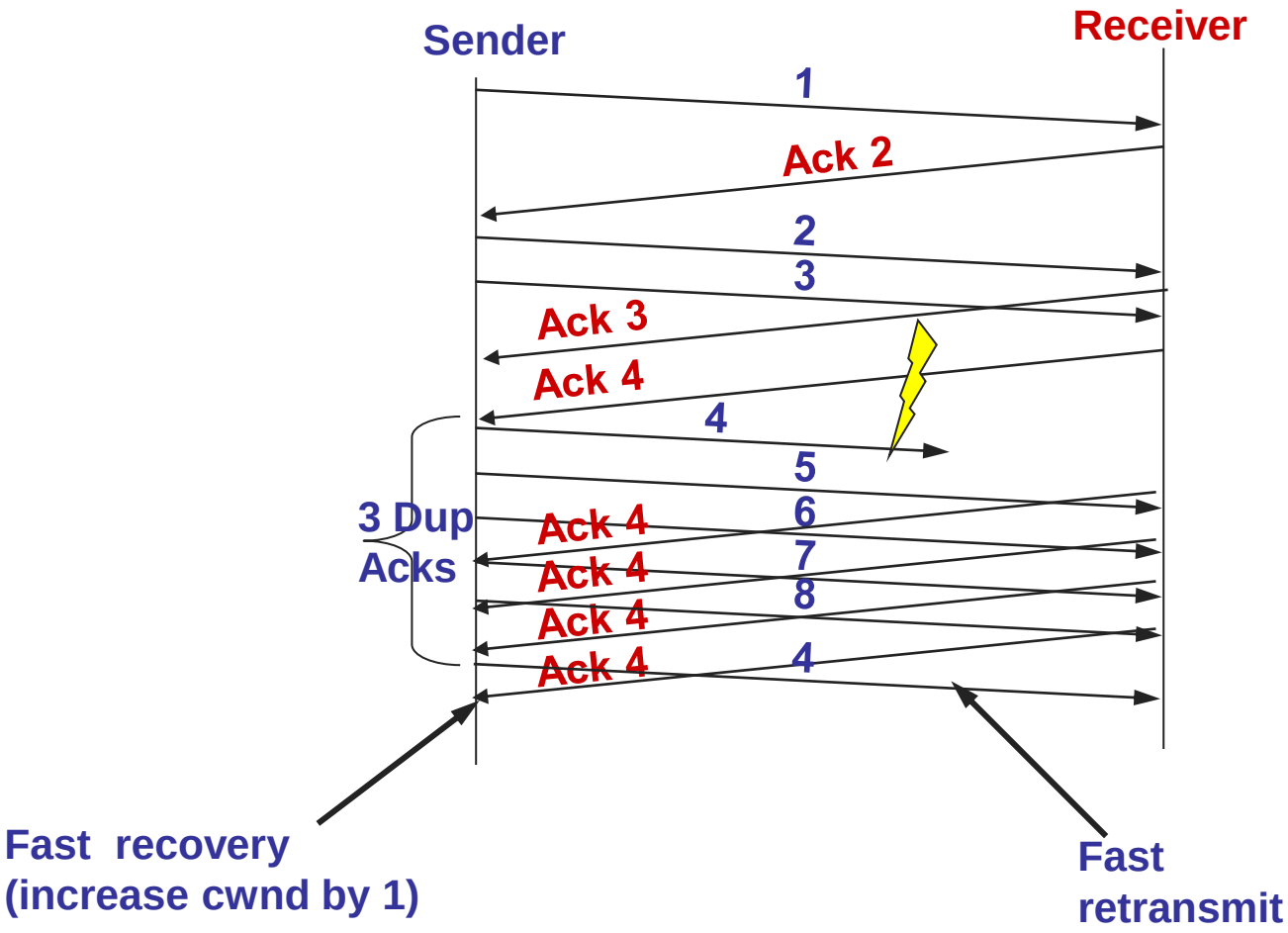


Fast Retransmit & Recovery

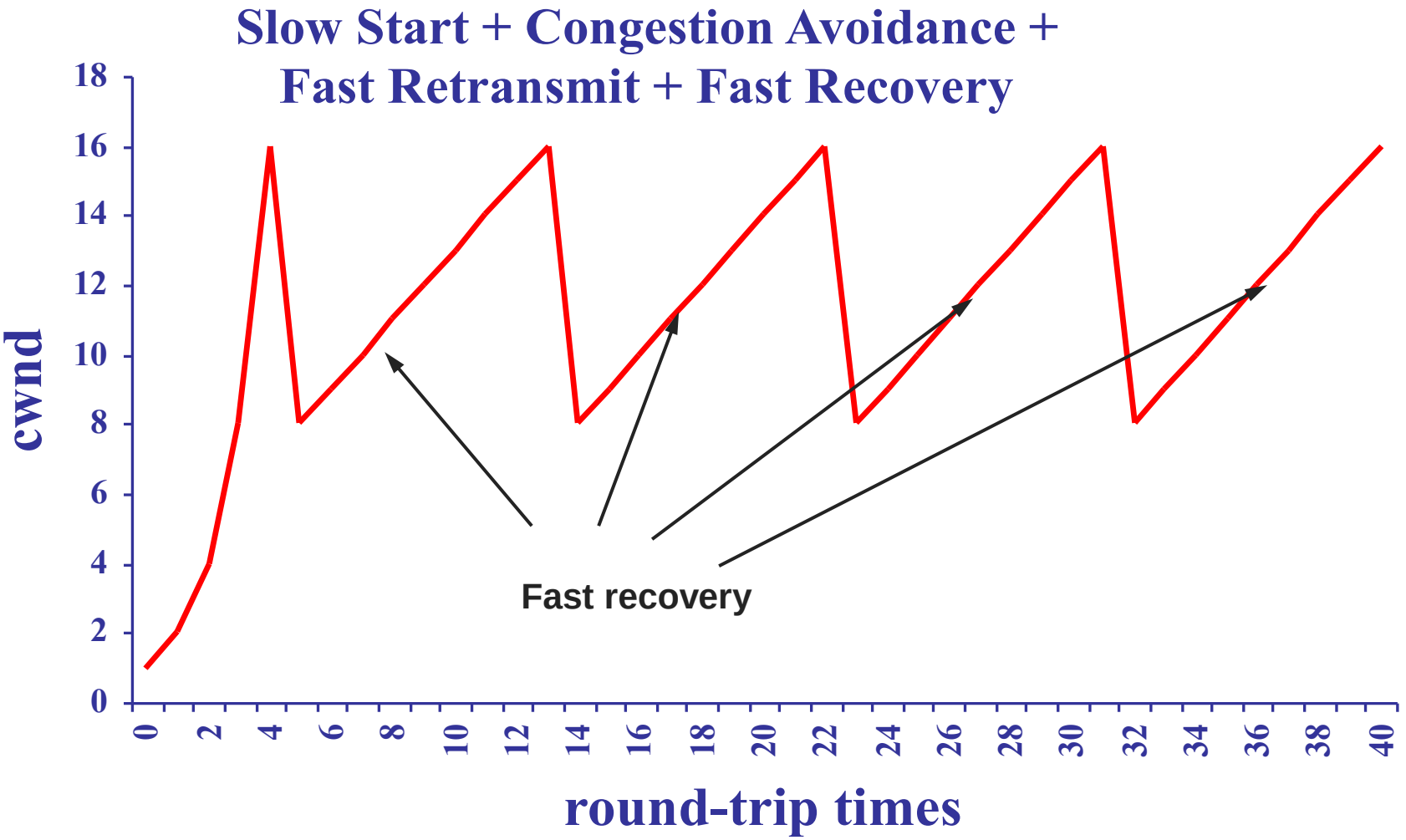
- Fast retransmit
  - Timeouts are slow (default often 200 ms or 1 second)
  - When packet is lost, receiver still ACKs last in-order packet
  - Use 3 duplicate ACKs to indicate a loss; detect losses quickly
    - » Why 3? When wouldn't this work?

- Fast recovery
  - Goal: avoid stalling after loss
  - If there are still ACKs coming in, then no need for slow start
  - If a packet has made it through -> we can send another one
  - Divide *cwnd* by 2 after fast retransmit
  - Increment cwnd by 1 MSS for each additional duplicate ACK

Fast Retransmit Example



More Sophistication



Short Connections

- Short connection: only contains a few pkts
- How do short connections and Slow-Start interact?
  - What happens when a packet is lost during Slow-Start?
  - What happens when the SYN is dropped?
- Bottom line: Which packet gets dropped matters a lot
  - SYN
  - Slow-Start
  - Congestion avoidance
- Do you think most flows are short or long?
- Do you think most traffic is in short flows or long flows?

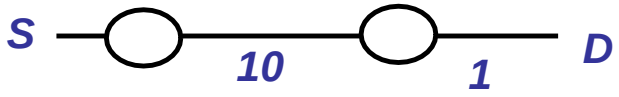
Open Issues

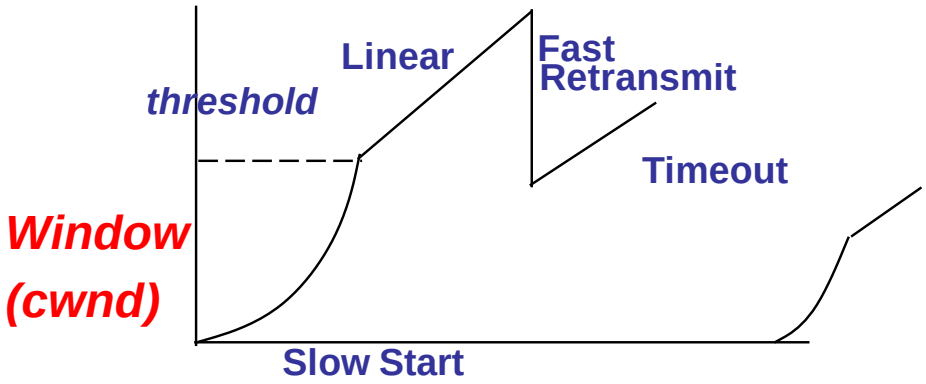
- TCP is designed around the premise of cooperation
  - What happens to TCP if it competes with a UDP flow?
  - What if we divide *cwnd* by 3 instead of 2 after a loss?
- There are a bunch of magic numbers
  - Decrease by 2x, increase by 1/*cwnd*, 3 duplicate acks, initial timeout = 3 seconds, etc.
- But overall it works really well!
  - Still being constantly tweaked…

TCP CC Summary

- TCP Probes the network for bandwidth, assuming that loss signals congestion
- The congestion window is managed with an additive increase/multiplicative decrease policy
  - It took fast retransmit and fast recovery to get there
  - Fast recovery keeps pipe “full” while recovering from a loss
- Slow start is used to avoid lengthy initial delays
  - Ramp up to near target rate, then switch to AIMD

TCP CONGESTION CONTROL SUMMARY





Part 2: TCP + Router Scheduling

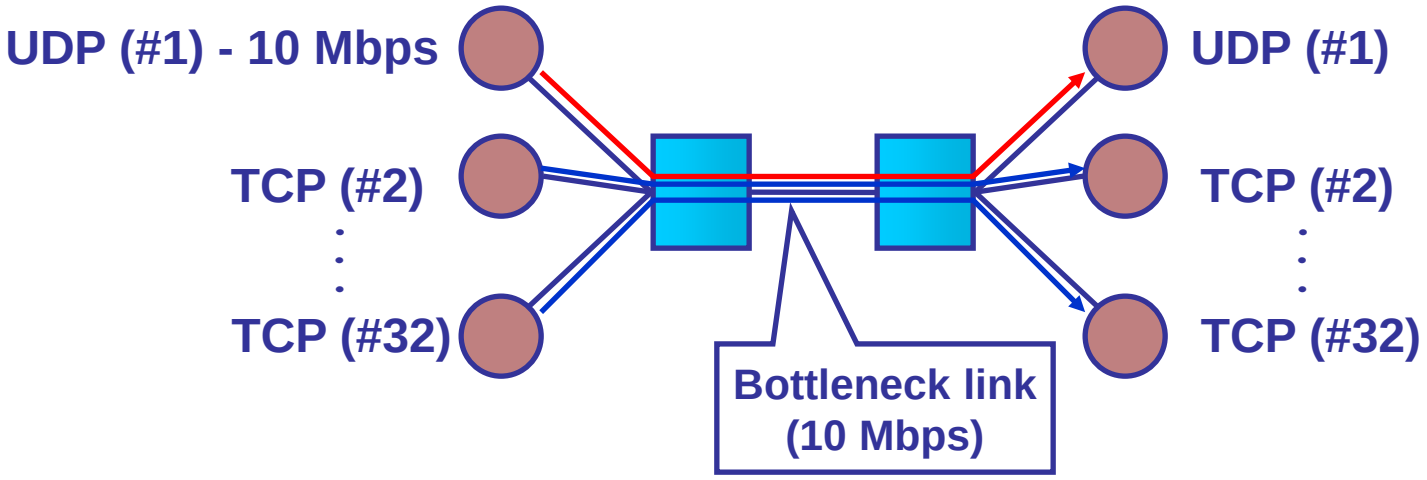
Help from routers (RED, ECN), TCP Fairness,
and some newer stuff (BBR, DCTCP)

TCP + Router Scheduling

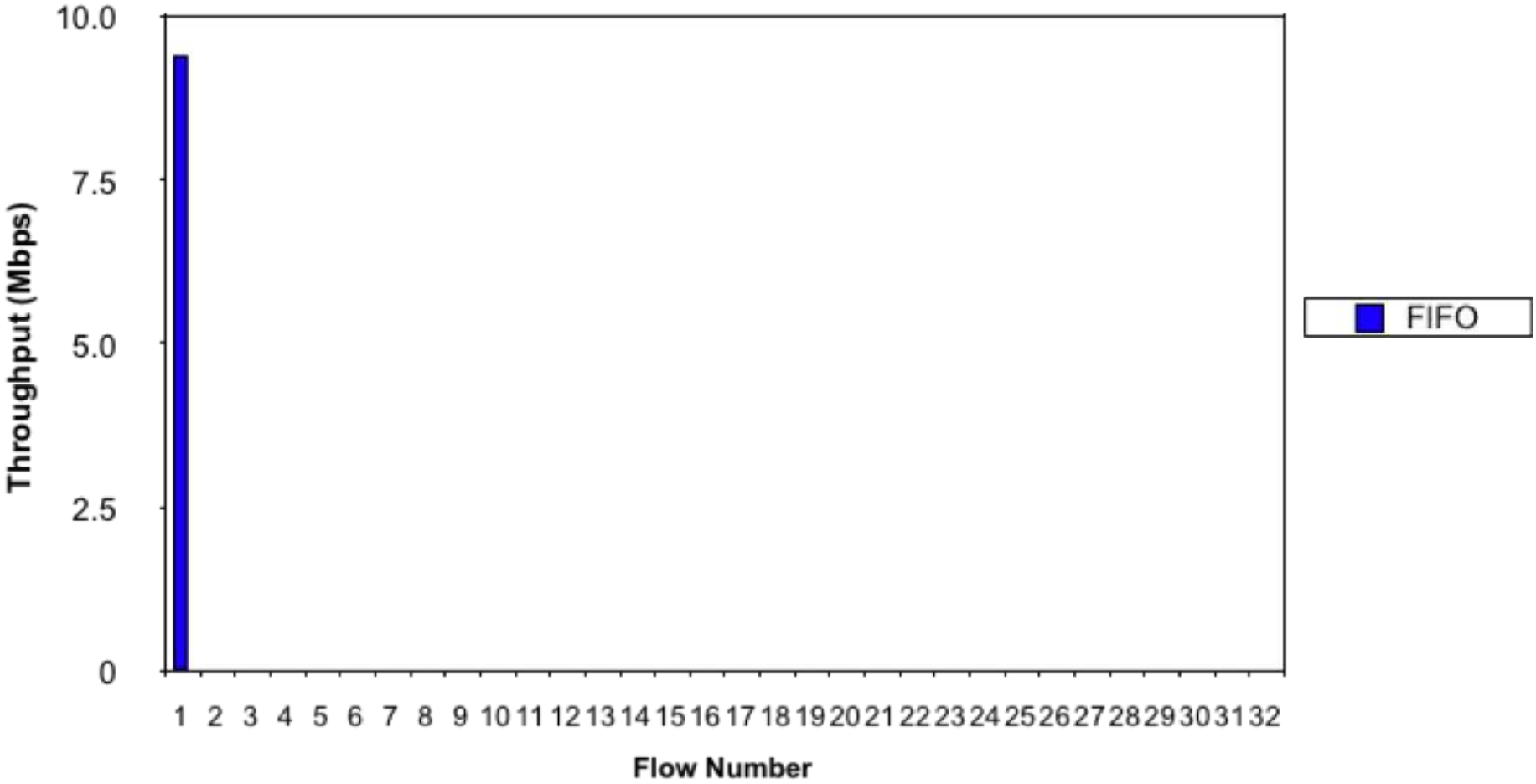
- So far we've done flow-based traffic policing
  - Limit the rate of one flow regardless of the load in the network
- In general, need scheduling at routers as well!
  - Dynamically allocate resources when multiple flows compete
  - Give each "flow" (or traffic class) own queue (at least theoretically)
- Weighted fair queuing
  - Schedule round-robins among queues in proportion to some weight parameter. Deficit Round Robin

Why we need router scheduling

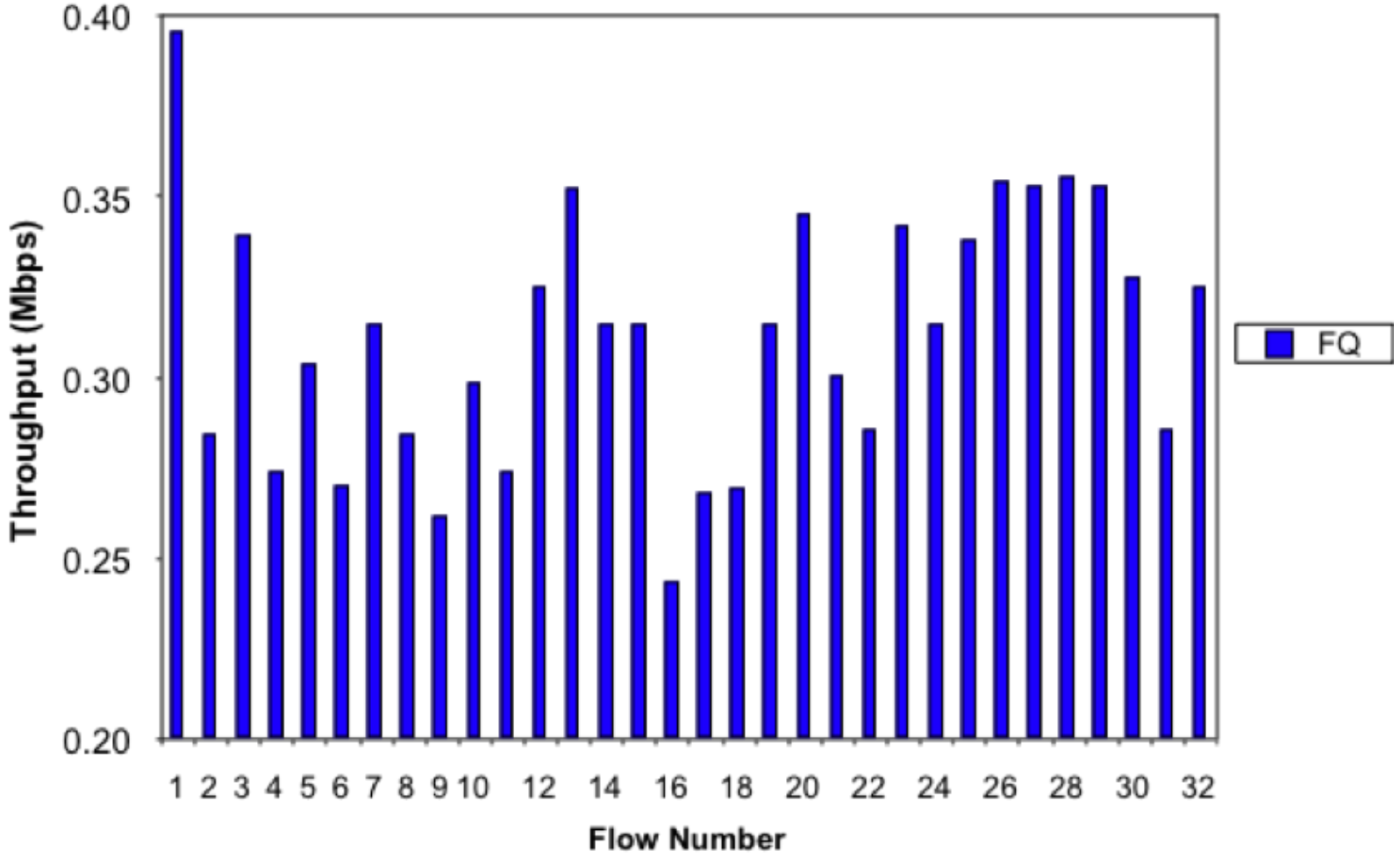
1 UDP (10 Mbps) and 31 TCPs sharing a 10 Mbps line



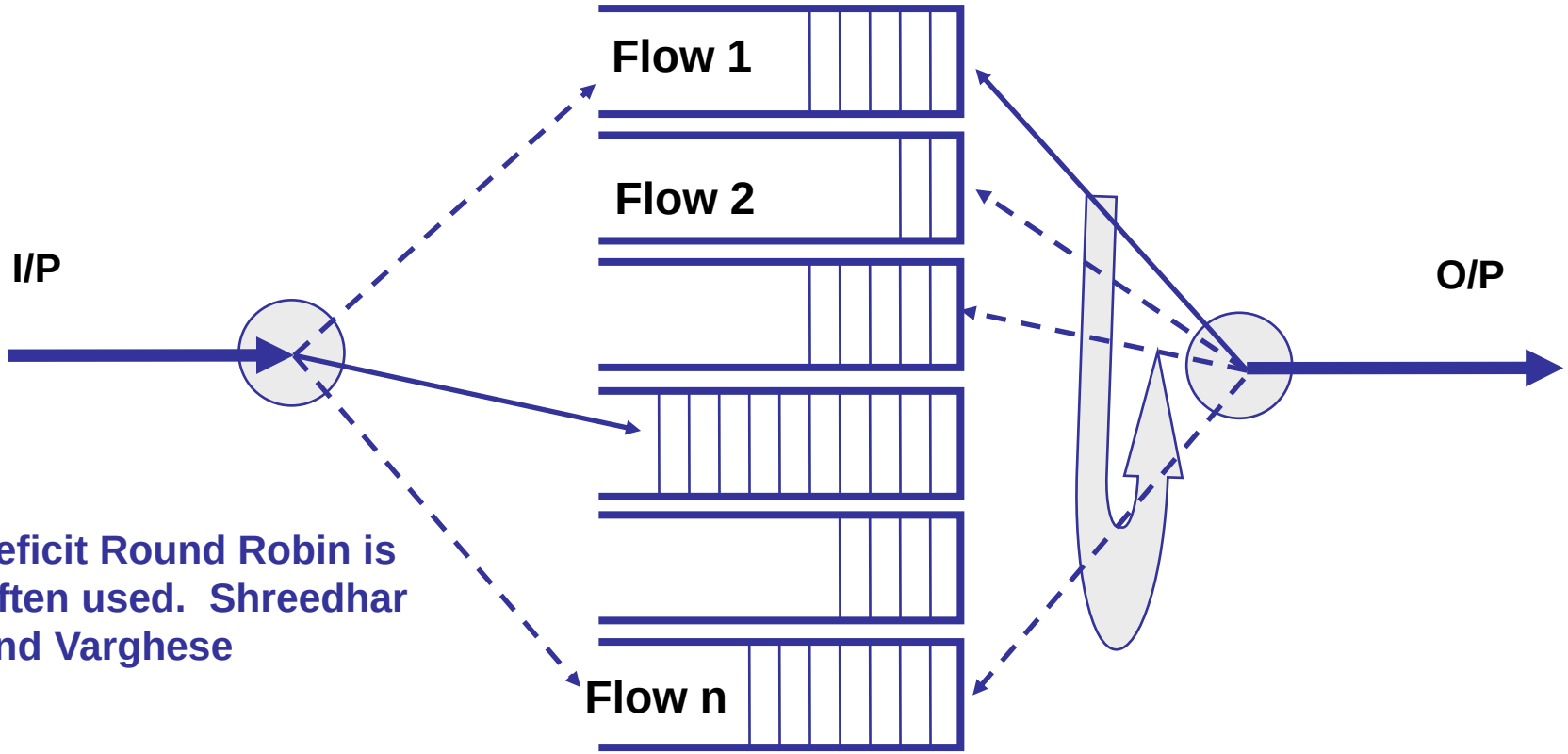
UDP *vs.* TCP w/FIFO



TCP vs. UDP w/Fair Queuing

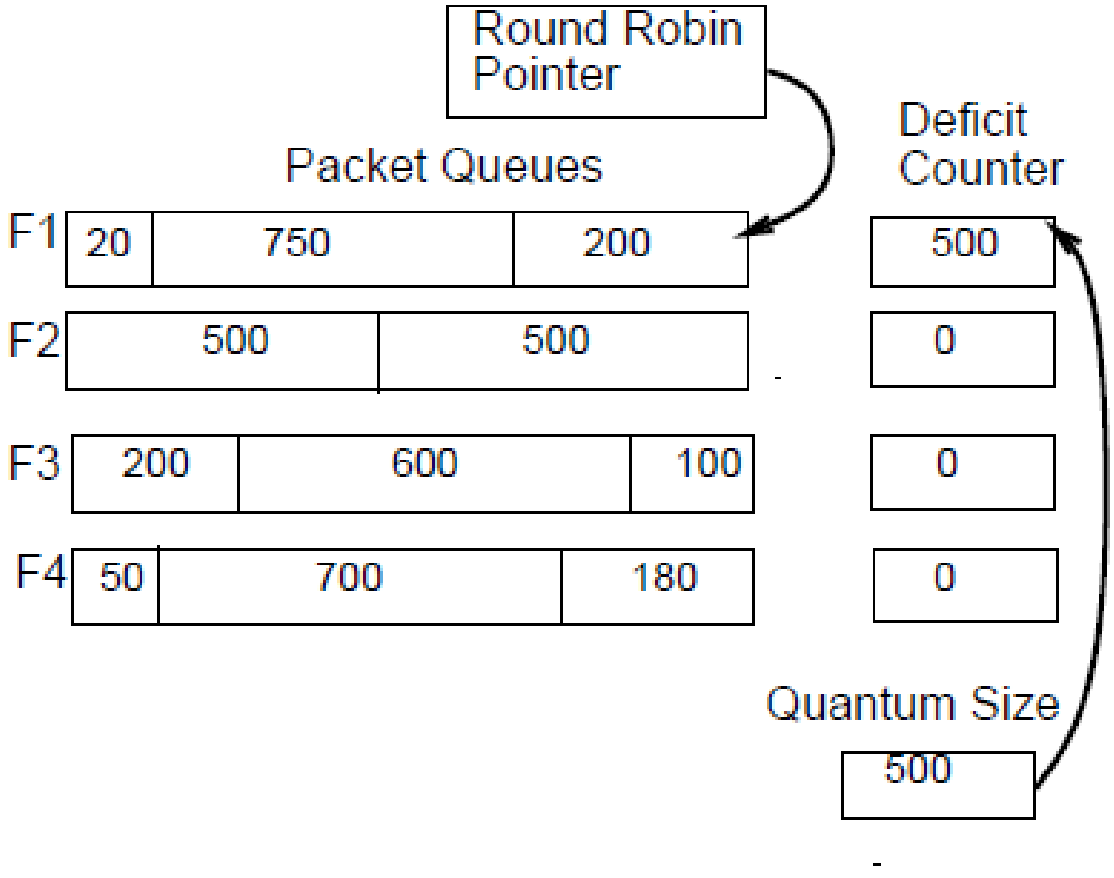


(Weighted) Fair Queuing

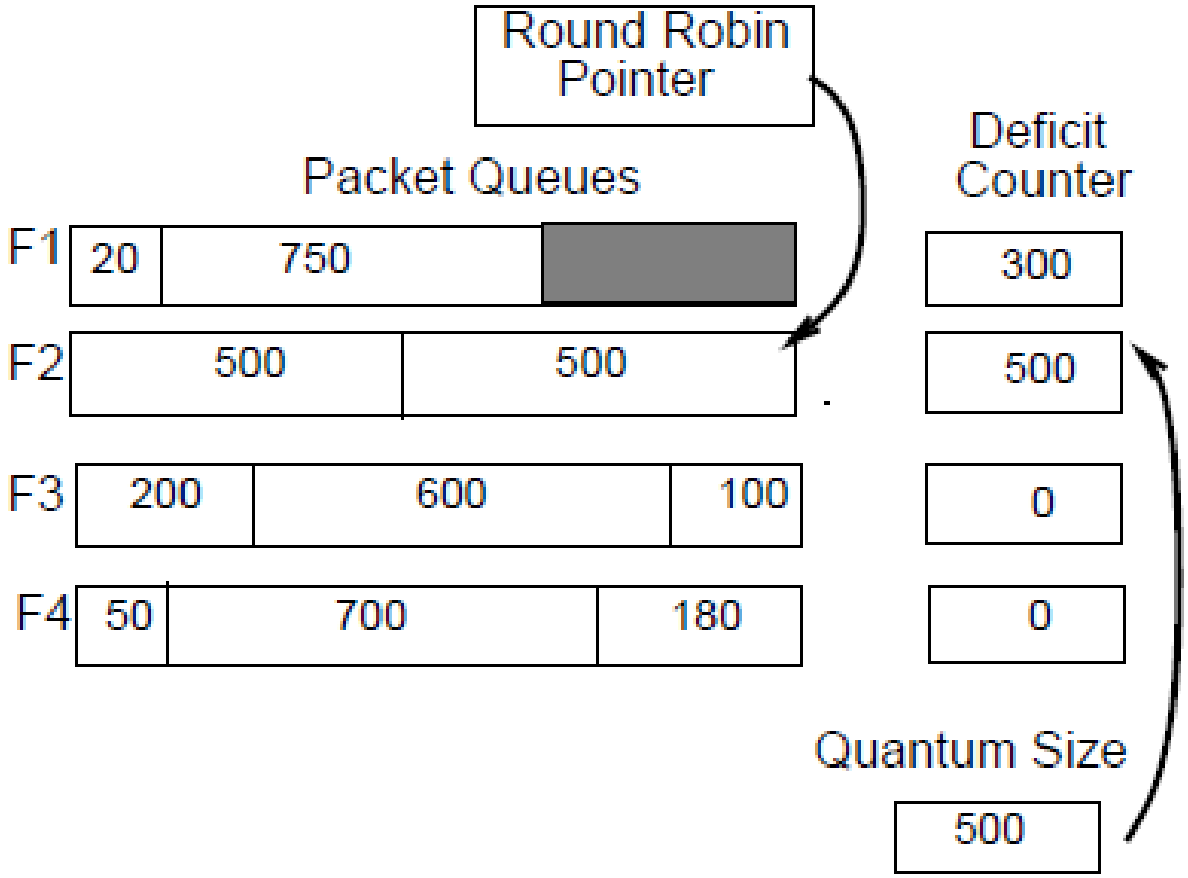


Deficit Round Robin is Often used. Shreedhar And Varghese

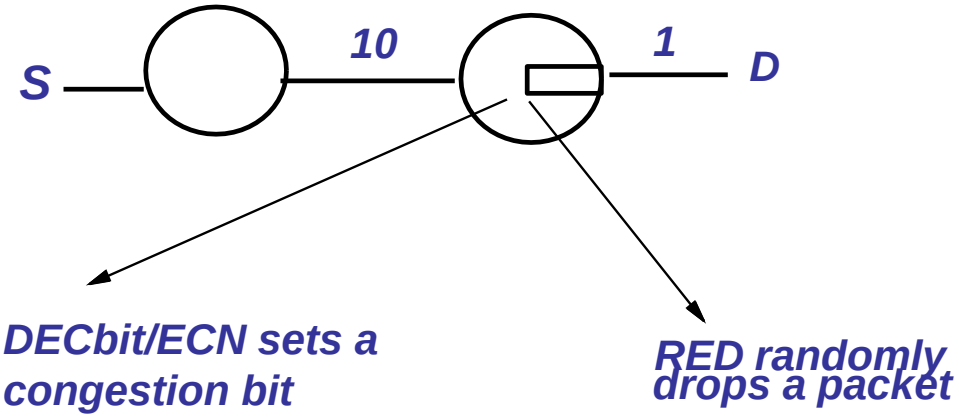
How DRR works



After sending packet

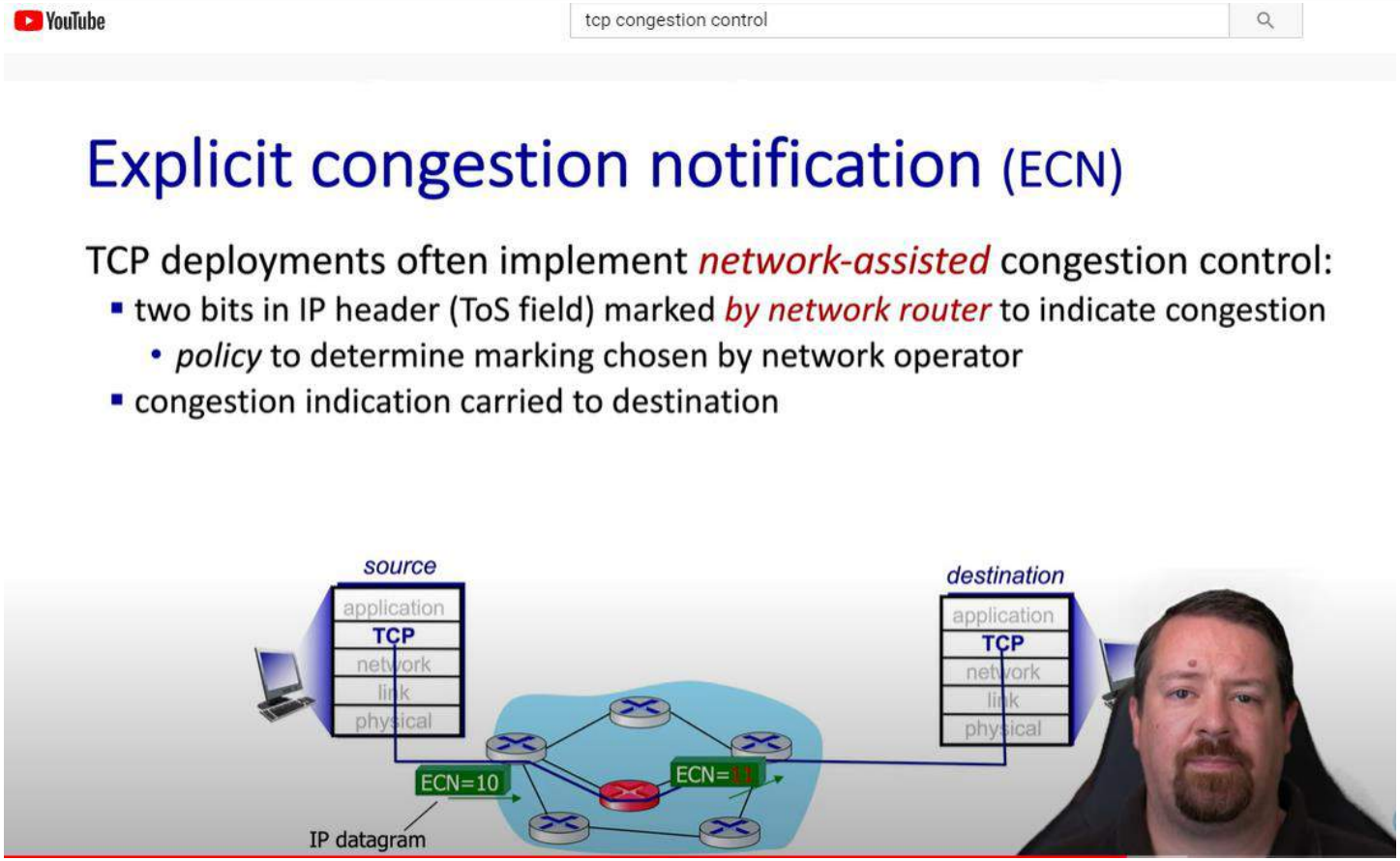


WHY ROUTERS ALSO DO RANDOM EARLY DETECT



Q: Why would a router drop a perfectly good packet even if has buffer space
A: As an early form of congestion warning if one does not have a congestion bit. Many IP routers have such a bit today, called the ECN bit

Summary + new ideas



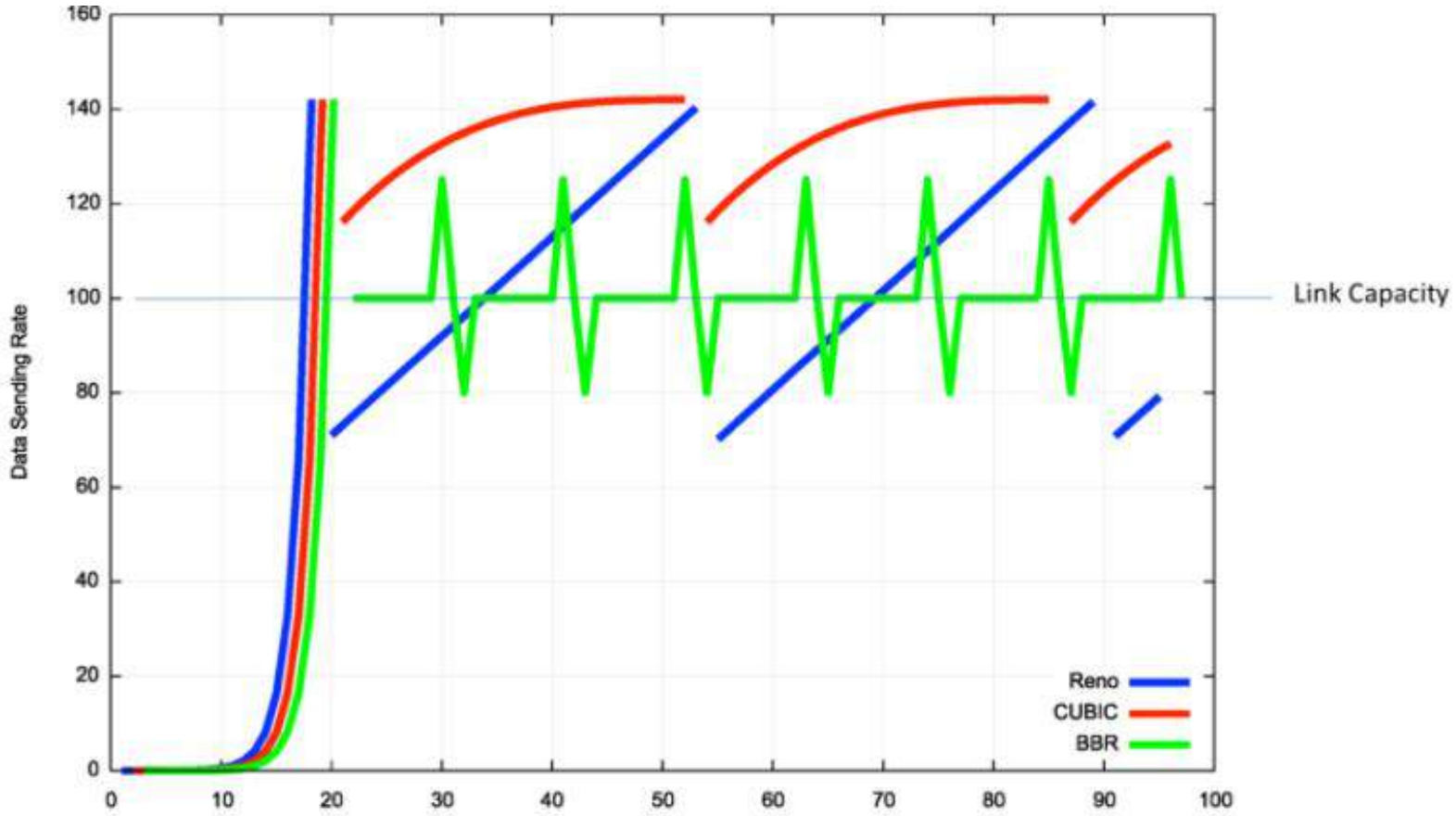
https://www.youtube.com/watch?v=rib_ujnMqcs

First 5 minutes for class
Next 5 for enrichment

New ideas in Youtube Video

- CUBIC: Increases window as cube of time from stable point. Used in Linux
- BBR: Delay based congestion control does not rely on loss so is smoother. Used within Google.
- ECN: Explicit congestion notification upon which TCP cuts window in half. Proactively avoids loss. Also hacked to do DCTCP (next slide)

Reno, CUBIC, BBR



https://blog.apnic.net/2017/05/09/bbr-new-kid-tcp-block/

Super tutorial: only for the curious!!! Not for exam

More new ideas

- TCP builds large queues. DCTCP uses ECN but counts how often ECN is set and so can cut window proportionately. Used in Microsoft, Apple DCs

- HTTP Pipelining: To reduce latency browser opens up multiple connections. Still slow over TCP.

- QUIC: Layered below HTTP and above TCP, places multiple streams in a single connection. Finesses slow start. Loss on one stream does not interrupt other stream. *Shares congestion among streams.*

Lecture Summary

- Unlike a reliable data link we also need to match speed of sender to receiver AND the whole network
- TCP increases and decreases its window using Slow Start + AIMD, decreasing on loss or ECN
- Needs to be accompanied by router scheduling mechanisms like DRR and RED/ECN setting
- TCP fluctuates too much. New protocols like DCTCP, Google's BBR adjust more smoothly