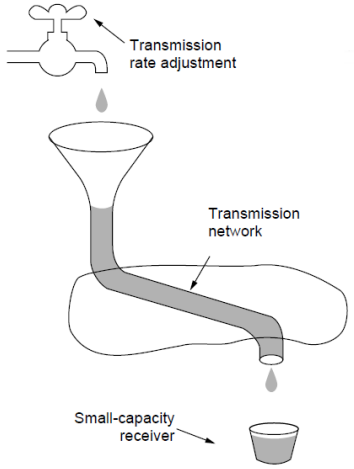
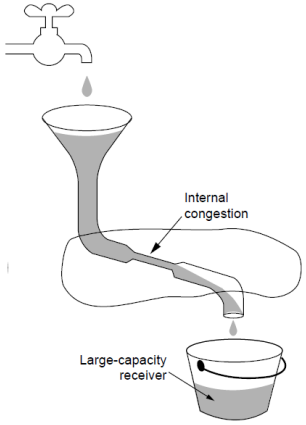
Congestion control

How is congestion related to sending rate?



A fast network feeding a low-capacity receiver → flow control is needed

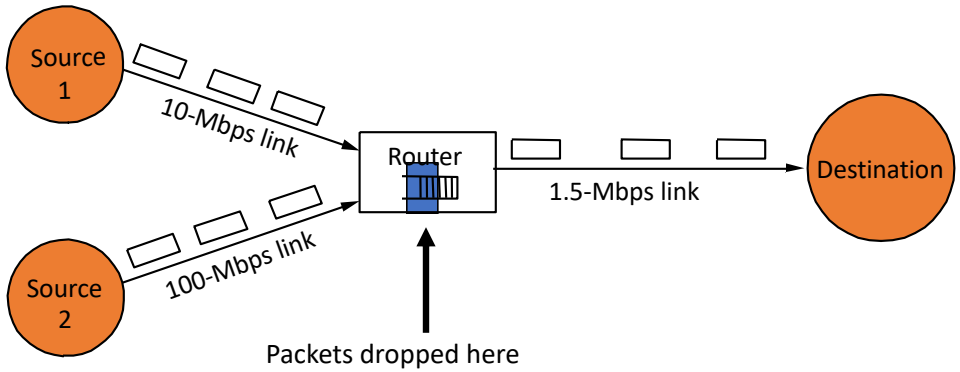


A slow network feeding a high-capacity receiver → congestion control is needed

What is congestion?

- All packets are stored in a router and then forwarded to the next hop. The router has a buffer of a certain size.
  - More details of this in the network layer
- If the processing time in the router >> incoming rate of packets
  - Congestion occurs and there is packet loss
- If you have too much congestion and retransmission, the network will collapse

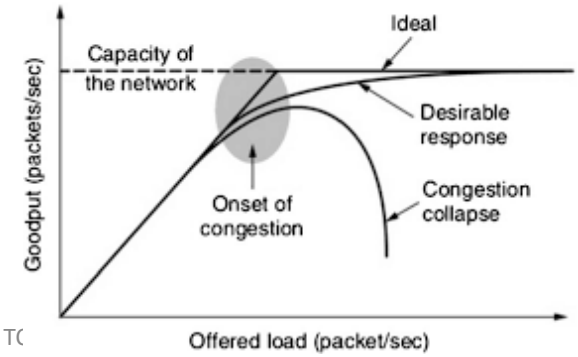
Why does this congestion collapse occur?



- 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

Congestion collapse in 1986

- An Internet congestion collapse was detected in October 1986 on a 32 kbps link between the University of California Berkeley campus and the Lawrence Berkeley National Laboratory that is 400 yards away, during which the throughput dropped by a factor of almost 1,000 to 40 bps.



Slight aside: Congestion and router queues

- Packet delay = queuing delay at the router + transmission delay (usually very small)

- If buffer size is high
  - Queuing delay is high
- If buffer sizes are small
  - Higher packet loss

- So there is no right buffer size to use at the routers that is good for the network

Solution: Congestion control

What should the window size be to not overwhelm the network?

- How to find if an internal router is congested?
  - There are many internal routers, the end hosts cannot poll each
  - What happens if new flows join the network and flows leave (not necessarily from the same host)?

- TCP follows the **end-to-end principle**
  - End hosts should not interact with routers at the transport layer
  - The end hosts should decide how much data to send to the network to avoid congestion at the routers

Challenges in bandwidth allocation for congestion control and fairness

- Who tells the end host
  - The available capacity or the operating point?
  - The presence of a bottleneck link?

- What happens if new flows join the network and flows leave (not necessarily from the same host)

- How to ensure fairness of flows that only have part overlapping paths?

Possible approaches

- Do nothing, send packets indiscriminately
  - Many packets will drop, totally unpredictable performance
  - May lead to congestion collapse
- Reservations
  - Pre-arrange bandwidth allocations for flows
  - Requires negotiation before sending packets
  - Must be supported by the network

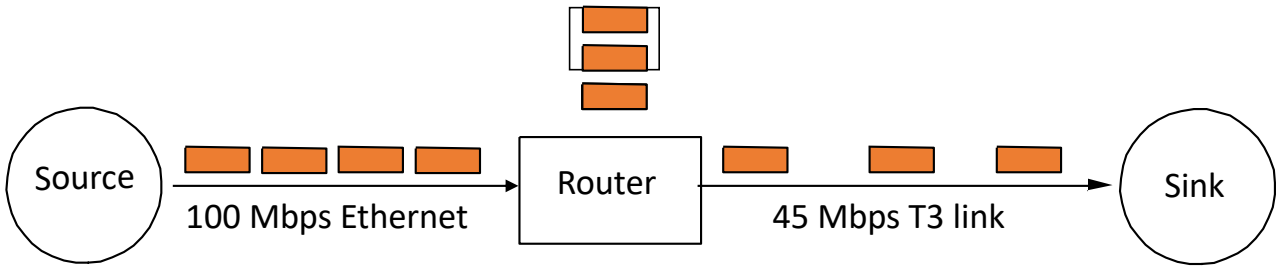
Instead TCP using dynamic adjustment

- Use probes to estimate level of congestion
- Speed up when congestion is low
- Slow down when congestion increases

TCP's End-to-End argument

- End host "learns" the rate at which it can pump data into the network using probes
- End host "learns" the existence of a congestion
- End host automatically adapts sending rate according to congestion

First probe the network, but start slow



- Each source independently probes the network by sending packets at a slow rate
- The rate is then adapted.

How to ``learn'' about bottlenecks?

- Implicit feedback: Losses = congestion
- Sending rate and losses are now intertwined
  - Increase the congestion window until there is a loss
  - Loss implies bottleneck, stop sending more data
- Reacting to losses also allows TCP to adjust as flows join or leave the network

AIMD: Additive Increase Multiplicative decrease

- Increase the sending rate slowly
- On loss, reduce the sending rate rapidly
- Sending rate == Window size.
  - Called the congestion window size of *cwnd*.

TCP congestion control idea

- Send congestion window size of packets (cwnd)
- Use AIMD to adapt cwnd
  - Increase cwnd slowly when bandwidth is available
  - Reduce cwnd rapidly when loss occurs.
- Use packet loss as a signal for congestion.

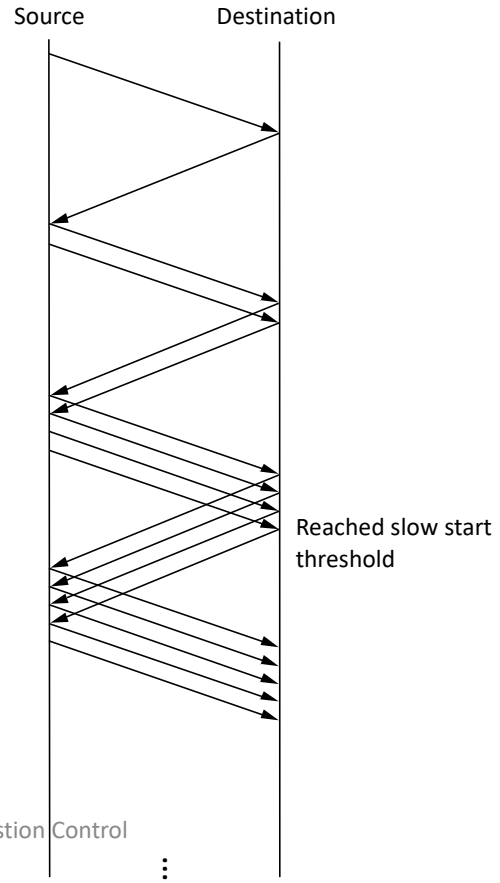
TCP congestion control algorithm:

- TCP start up problem
  - Starting very slowly can waste bandwidth
  - Starting too quickly can cause congestion

- Solution: Start quickly up to a point (slow start phase), and then go slow (congestion avoidance phase).

TCP “Slow Start”

- Until slow start threshold
  - Double cwnd every RTT
  - Cwnd *= 2 / RTT
- When the slow start threshold is reached, start additive increase
  - Cwnd +=1 / packet received



TCP congestion control algorithm: Tahoe(1 of 2)

- Slow start phase (actually not slow)
  - First send a small number of packets == initial congestion window size (icwnd)
  - For every RTT, double cwnd (i.e., for every ack send an additional packet)
  - (If cwnd >= slow start threshold *ssthresh*)
    - Go to congestion avoidance phase.
  - If there is a loss
    - cwnd = icwnd, ssthresh = cwnd/2.

TCP congestion control algorithm: Tahoe (2 of 2)

- Congestion avoidance phase
  - Send cwnd packets
  - For every RTT, increase cwnd by 1 (i.e., for every ack, increase cwnd by 1/cwnd)
  - When timeout occurs
    - cwnd = icwnd, ssthresh = cwnd/2
- This version is called TCP Tahoe.

But timeout is very expensive: Instead, use Fast Retransmit

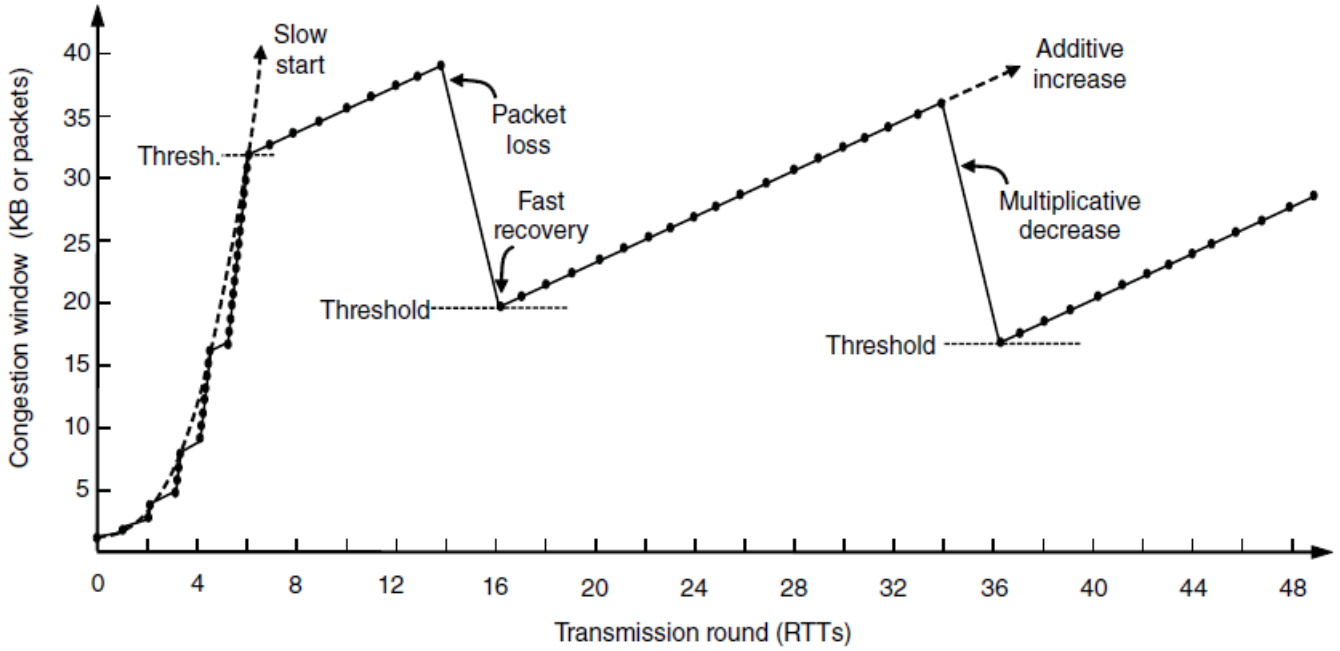
- Fast Retransmit
  - Duplicate acks are a signal that a packet is lost
  - When triple duplicate acks arrive, assume that the packet is lost and retransmit the packet
  - TCP Reno is a congestion control algorithm that improves TCP Tahoe using fast retransmits

TCP Reno: Fast Retransmit & Fast Recovery

- Congestion avoidance stage
  - When timeout occurs, same as TCP Tahoe
  - But if loss is indicated through triple duplicate ack
    - Resend lost packet
    - ssthresh = cwnd/2, cwnd = cwnd/2 (remain in congestion avoidance)

- Slow start
  - When timeout occurs, same as TCP Tahoe
  - On triple duplicate ack
    - Resend lost packet
    - ssthresh = cwnd/2, cwnd = cwnd/2 (move directly from slow start to congestion avoidance)

TCP Reno



Assume no timeout!

Why is congestion control hard in practice?

- TCP Performs poorly in wireless networks where losses are not due to congestion

- TCP performs poorly when the bandwidth is large and the RTT is large
  - Because a lot more packets can fit in the pipe but TCP is too slow

- TCP performs poorly in datacenters that require extremely small delays

In general, TCP infers the bottleneck from the network and this inference can go wrong.

Many variants of congestion control

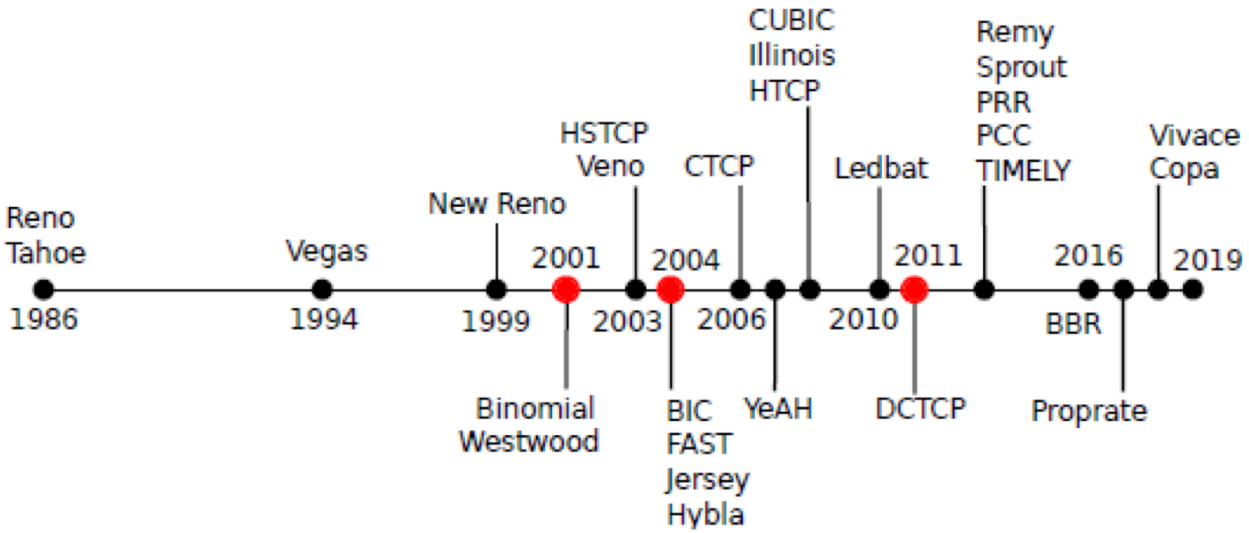
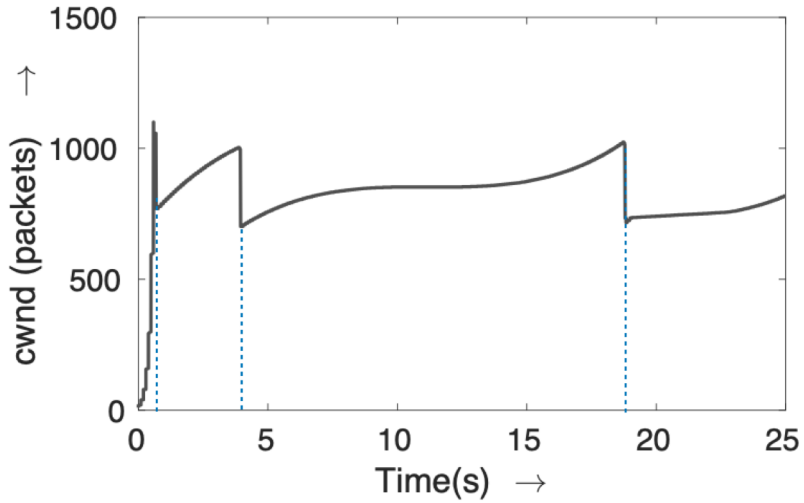


Fig. 1. The evolution of TCP congestion control.

- TCP congestion control is one of the most researched areas in networking

TCP loss-based variants

- TCP Cubic: Instead of AIMD, the increase and decrease function follow a cubic formula. **This is default on most Linux and MAC OSes**



Other TCP variants (1 of 2)

- Delay based: TCP Vegas
  - Instead of using loss as the indicator for congestion uses delay as an indicator for congestion
  - The default on Windows is Compound TCP that is a combination of delay and loss based
- Datacenters: DCTCP
  - Uses a congestion indicator called Explicit Congestion Notification (ECN) from the router

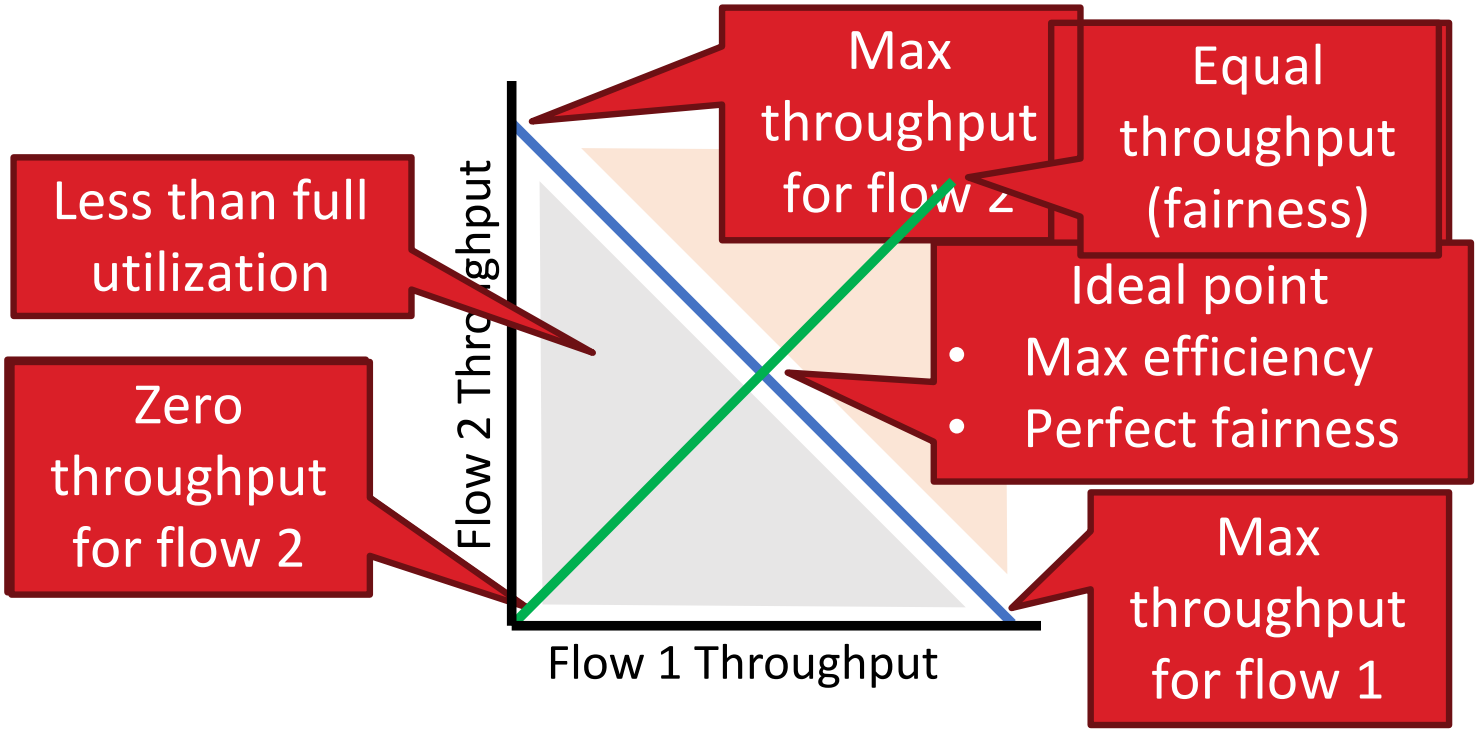
Other TCP variants (2 of 2)

- Newest TCP variant: BBR from Google
  - This solves the problem of adjusting quickly to large RTT and large bandwidth
  - Randomly probes the network for more bandwidth
- Learning based Congestion control
  - COPA, etc

Other TCP enhancements

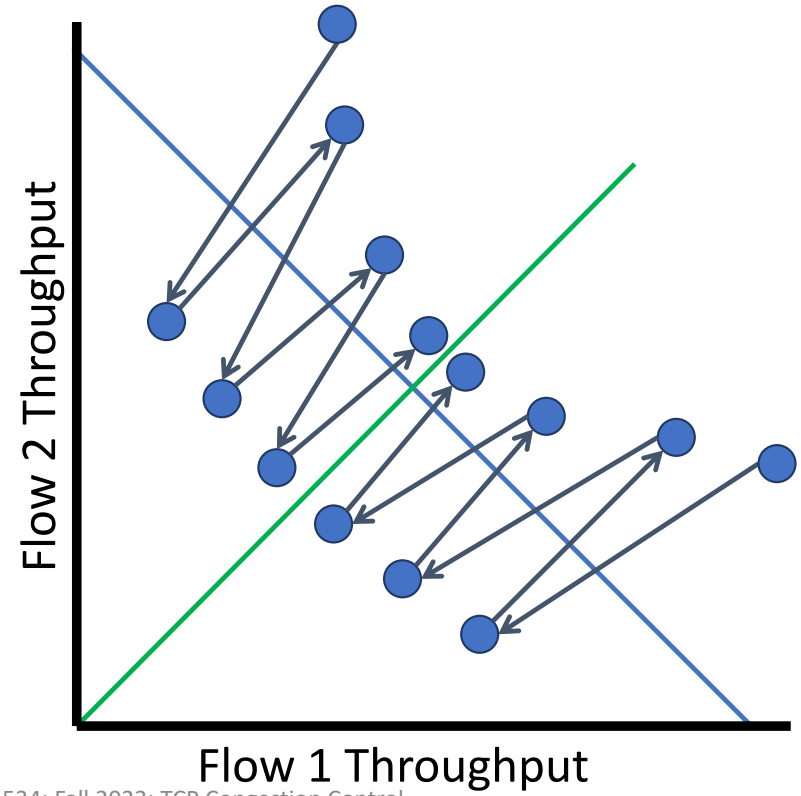
- Initial congestion window size
  - Google uses 32 as a icwnd value
- Router enhancements
  - Random Early Detection (RED)
  - Random Exponential Marking using ECN (used in datacenters)
- Ack enhancements
  - SACK, NACK

Utilization and Fairness



Additive Increase, Multiplicative Decrease

- Converges to stable and fair cycle
- Symmetric around *y=x*



What factors does TCP throughput depend on?

- RTT
- Loss rate
- Maximum segment size

Throughput = (√(3/2)* MSS)/(RTT * √p)

RTT = round trip time, p = Probability that a packet is lost, MSS = Maximum segment size in bytes