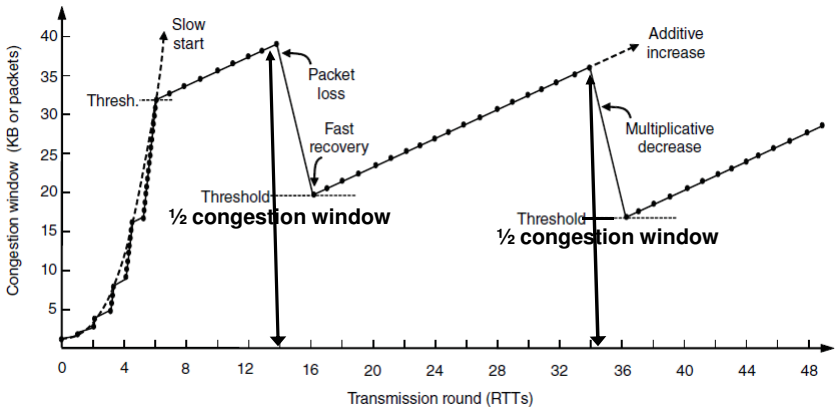
Recap



More TCP

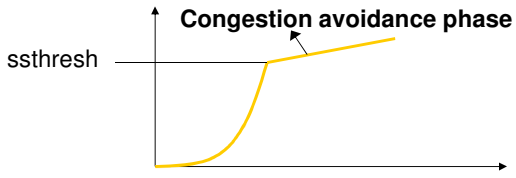
- Congestion avoidance
- TCP timers
- TCP lifeline

Application
Presentation
Session
Transport
Network
Data Link
Physical

Congestion Control vs Avoidance

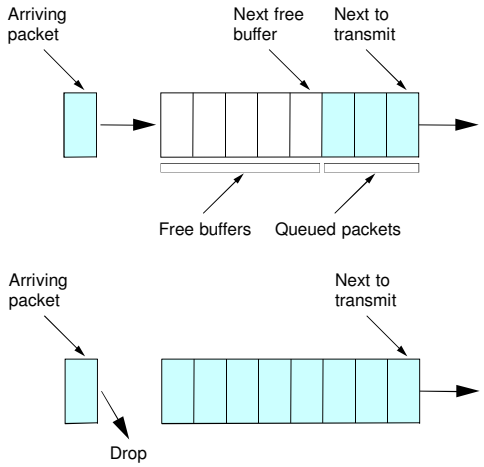
- TCP causes congestion as it probes for the available bandwidth and then recovers from it after the fact
  - Leads to loss, delay and bandwidth fluctuations
  - We want congestion avoidance, not congestion control
- Congestion avoidance mechanisms
  - Aim to detect incipient congestion, before loss. So monitor queues to see that they absorb bursts, but not build steadily

TCP protocol uses some kind of avoidance

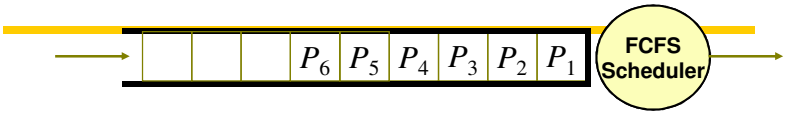


- Avoid congestion by increasing linearly
- Can we do more?

Router Model: "FCFS with Tail Drop"

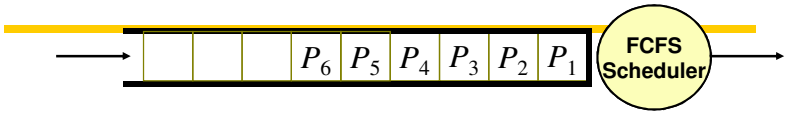


The case against drop-tail queue management



- Large queues in routers is "a bad thing"
  - Delay: end-to-end latency dominated by length of queues at switches in network
- Allowing queues to overflow is "a bad thing"
  - Fairness: connections transmitting at high rates can starve connections transmitting at low rates

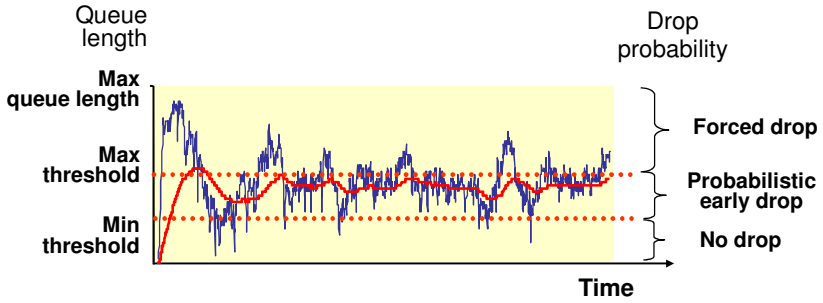
Random early packet drop (RED)



When queue length exceeds threshold, drop packets with queue length dependent *probability*

- probabilistic packet drop: flows see same loss *rate*
- problem: bursty traffic (burst arrives when queue is near threshold) can be over penalized

Random early detection (RED) packet drop



- Use exponential *average* of queue length to determine when to drop
  - avoid overly penalizing short-term bursts
  - react to longer term trends

RED summary: why random drop?

- Provide gentle transition from no-drop to all-drop
  - Provide "gentle" early warning
  - Avoid synchronized loss bursts among sources
- Provide same loss rate to all sessions:
  - With tail-drop, low-sending-rate sessions can be completely starved

Explicit congestion notification

- Can we avoid congestion without loss?
- Can the routers signal the hosts (this is a bit off from the end-to-end argument)
  - Do not want to send additional packets.

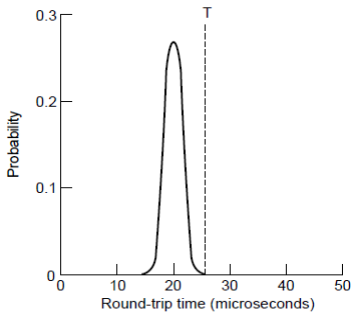
Explicit Congestion Notification (ECN)

- ECN signals congestion with a bit in the IP header
- Receiver returns indication to the sender, who slows
  - Need to signal this reliably or we risk instability
- Network-assisted congestion control

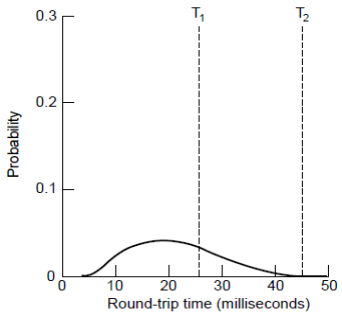
Deciding When to Retransmit

- How do you know when a packet has been lost?
  - Ultimately sender uses timers to decide when to retransmit
- But how long should the timer be?
  - Too long: inefficient (large delays, poor use of bandwidth)
  - Too short: may retransmit unnecessarily (causing extra traffic)
  - A good retransmission timer is important for good performance
- Right timer is based on the round trip time (RTT)
  - Which varies greatly in the wide area. Why?

RTT variance in LANs versus Internet



LAN case – small, regular RTT



Internet case – large, varied RTT

Congestion Collapse due to incorrect RTT estimates

- In the limit, early retransmissions lead to congestion collapse
  - Sending more packets into the network when it is overloaded exacerbates the problem of congestion
  - Network stays busy but very little useful work is being done
- This happened in real life ~1987
  - Led to Van Jacobson's TCP algorithms, which form the basis of congestion control in the Internet today

[See "Congestion Avoidance and Control", SIGCOMM'88]

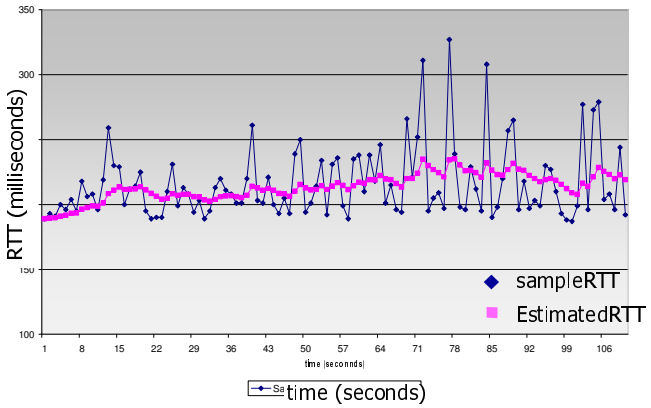
Estimating RTTs

- Idea: Adapt retransmission timer based on recent past measurements

- Simple algorithm:
  - For each packet, note time sent and time ack received
  - Compute RTT samples and average recent samples for timeout
  - EstimatedRTT = α x EstimatedRTT + (1 - α) x SampleRTT
  - This is an exponentially-weighted moving average (low pass filter) that smoothes the samples. Typically, α = 0.8 to 0.9.
  - Set timeout to small multiple (2) of the estimate

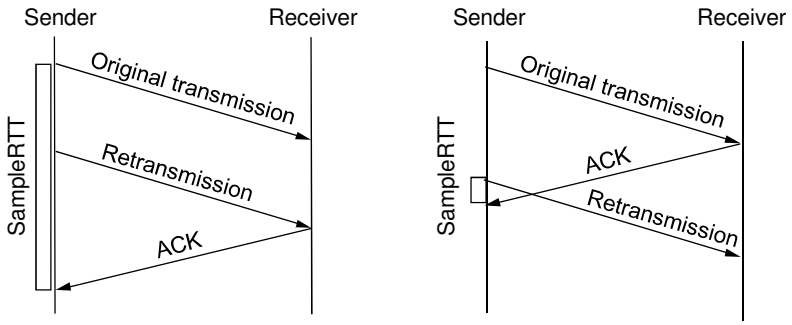
TCP round trip time, timeout

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT



Karn/Partridge Algorithm

- Problem: RTT for retransmitted packets ambiguous



- Solution: Don't measure RTT for retransmitted packets and do not relax backed of timeout until valid RTT measurements

Jacobson/Karels Algorithm

- Problem:
  - Variance in RTTs gets large as network gets loaded
  - So an average RTT isn't a good predictor when we need it most
- Solution: Track variance too.
  - Difference = SampleRTT – EstimatedRTT
  - EstimatedRTT = EstimatedRTT + (δ x Difference)
  - Deviation = Deviation + δ(|Difference|- Deviation)
  - Timeout = μ x EstimatedRTT + ϕ x Deviation
  - In practice, δ = 1/8, μ = 1 and ϕ = 4

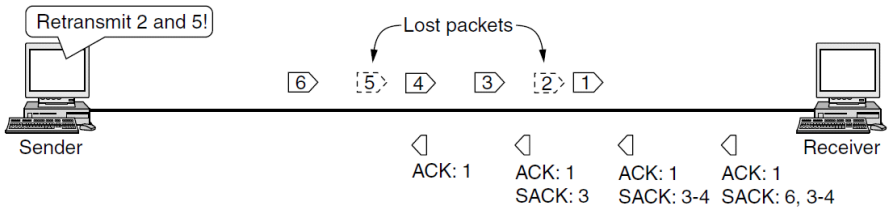
So far we saw Loss-based TCP

- Evolution of loss-based TCP
  - Tahoe
  - Reno
  - Selective Acknowledgment (explained in next slide)
- Q: what if loss not due to congestion?

Selective ACKS

Extend ACKs with a vector to describe received segments and hence losses

- Allows for more accurate retransmissions / recovery



No way for us to know that 2 and 5 were lost with only ACKs

Delay-based TCP Vegas

- Uses delay as a signal of congestion
  - Idea: try to keep a small constant number of packets at bottleneck queue
  - Expected = W/BaseRTT
  - Actual = W/CurRTT
  - Diff = Expected - Actual
  - Try to keep Diff small
- Delay-based TCP not widely used today

Wireless Issues

- Wireless links lose packets due to transmission errors
  - Do not want to confuse this loss with congestion
  - Or connection will run slowly over wireless links!
- One Strategy:
  - Wireless links use ARQ, which masks errors