CSE 561 - Lectures 6-7

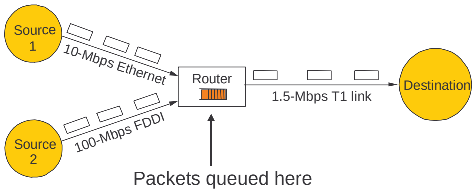
Congestion Control

David Wetherall
djw@cs.washington.edu

Deciding When to Retransmit

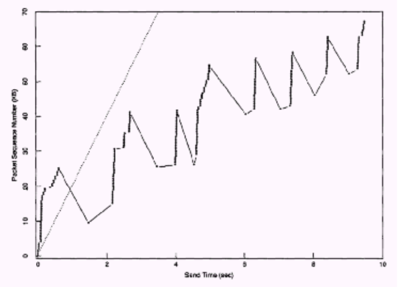
- How do you know when a packet has been lost?
  - Ultimately sender uses timers to decide when to retransmit
- But how long should the timer be?
  - Too long: inefficient (large delays, poor use of bandwidth)
  - Too short: may retransmit unnecessarily (causing extra traffic)
  - A good retransmission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

A Simple Network Model



- Buffers at routers used to absorb bursts when input rate > output
- Loss (drops) occur when sending rate is persistently > drain rate

Effects of Early Retransmissions



Congestion Collapse

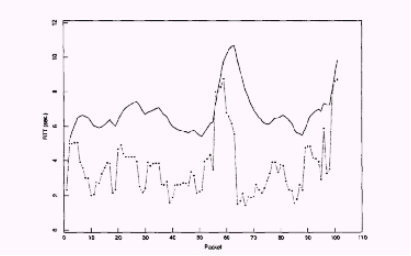
- In the limit, early retransmissions lead to congestion collapse
  - Sending more packets into the network when it is overloaded exacerbates the problem of congestion
  - Network stays busy but very little useful work is being done
- This happened in real life ~1987
  - Led to Van Jacobson's TCP algorithms,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

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

Estimating RTTs

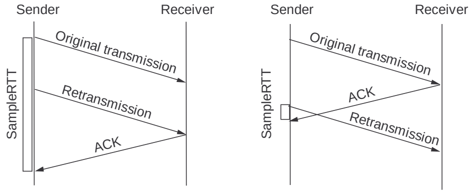
- Idea: Adapt based on recent past measurements
- Simple algorithm:
  - For each packet, note time sent and time ack received
  - Compute RTT samples and average recent samples for timeout
  - EstimatedRTT = α x EstimatedRTT + (1 - α) x SampleRTT
  - This is an exponentially-weighted moving average (low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

Estimated Retransmit Timer



Karn/Partridge Algorithm

- Problem: RTT for retransmitted packets ambiguous

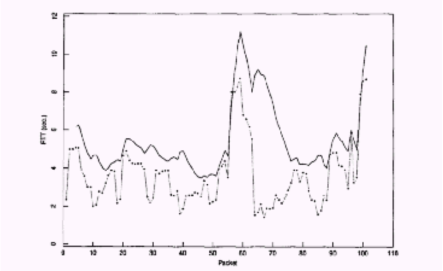


- Solution: Don’t measure RTT for retransmitted packets and do not relax backed of timeout until valid RTT measurements

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

Estimate with Mean + Variance



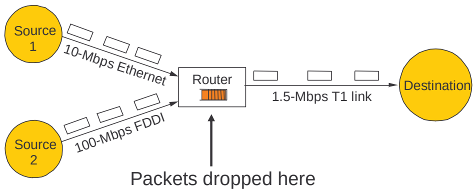
Key Concepts so Far

- A good retransmit timer is important for good performance
  - Too long leads to poor performance
  - Too short leads to wasted bandwidth
- An estimated timeout must adapt to Internet queuing
  - High variance at high load

Bandwidth Allocation

- How fast should the Web server send packets?
- Two big issues to solve!
- Congestion
  - sending too fast will cause packets to be lost in the network
- Fairness
  - different users should get their fair share of the bandwidth
- Often treated together (e.g. TCP) but needn’t be

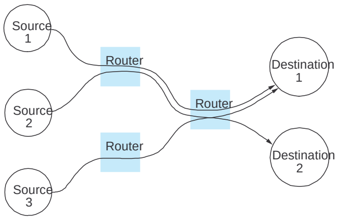
Congestion



Packets dropped here

- 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

Fairness



- Each flow from a source to a destination should get an equal share of the bottleneck link … depends on paths and other traffic

Bandwidth Allocation Approaches

- Open versus Closed loop
  - Open: reserve allowed traffic with network; avoid congestion
  - Closed: use network feedback to adjust sending rate
- Host-based versus Network support
  - Who is responsible for adjusting/enforcing allocations?
- Window versus Rate based
  - How is allocation expressed? Window and rate are related.

- Internet depends on TCP for bandwidth allocation
  - TCP is a host-driven, window-based, closed loop mechanism

Design Choices

- TCP/Internet provides “best-effort” service
  - Implicit network feedback, host controls via window.
  - No strong notions of fairness

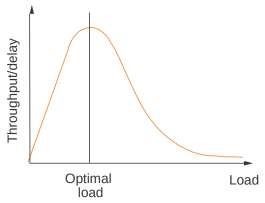
- A network in which there are QOS (quality of service) guarantees
  - Rate-based reservations natural choice for some apps
  - But reservations are need a good characterization of traffic
  - Network involvement typically needed to provide a guarantee

- Former tends to be simpler to build, latter offers greater service to applications but is more complex.

Evaluating Congestion Control

- Power = throughput / delay

- At low load, throughput goes up and delay remains small
- At moderate load, delay is increasing (queues) but throughput doesn’t grow much
- At high load, much loss and delay increases greatly due to retransmissions

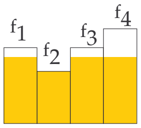


Evaluating Fairness

- First, need to define what is a fair allocation
  - Consider n flows, each wants a fraction f_i of the bandwidth

- Min-max fairness:
  - First satisfy all flows evenly up to the lowest f_i. Repeat with the remaining bandwidth.

- Also proportional fairness
  - Depends on path length …



Jain’s Fairness Index

- How do we compute the fairness of an allocation?
  - If all flows have an equal share at a router it’s “fair”
  - But how unfair are unequal allocations?
- Jain’s fairness index:
  - For n flows each receiving a fraction f_i of the bandwidth
  - Fairness = $(\Sigma f_i)^2 / (n \text{ x } \Sigma f_i^2)$
  - Always between 0 and 1, 1 for equal allocations
  - If only k out of n flows get bandwidth, drops to k/n

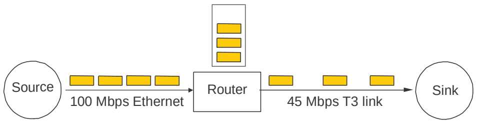
More Key Concepts

- Network mechanisms for bandwidth allocation should avoid congestion and provide fairness
- Congestion occurs when buffers inside the network fill with excess traffic
  - Queuing leads to increased latency and eventually to loss
- Fairness means that competing traffic flows gain a “fair share” of the available bandwidth
  - Min-max fairness is one definition of “fair share”

TCP Before Congestion Control

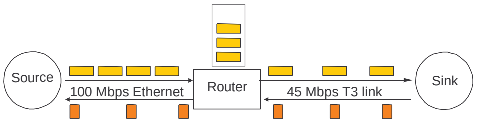
- Just use a fixed size sliding window!
  - Will under-utilize the network or cause unnecessary loss
- Congestion control dynamically varies the size of the window to match sending and available bandwidth
  - Sliding window uses minimum of cwnd, the congestion window, and the advertised flow control window
- The big question: how do we decide what size the window should be?

TCP Probes the Network



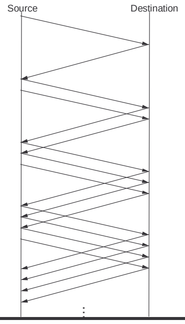
- Each source independently probes the network to determine how much bandwidth is available
  - Changes over time, since everyone does this
- Assume that packet loss implies congestion
  - Since errors are rare; also, requires no support from routers

TCP is “Self-Clocking”



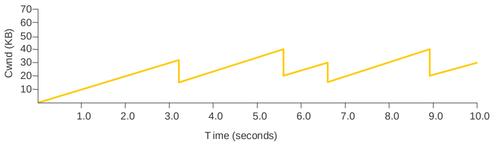
- Neat observation: acks pace transmissions at approximately the botteneck rate
- So just be sending packets we can discern the “right” sending rate (called the packet-pair technique)

AIMD (Additive Increase/Multiplicative Decrease)



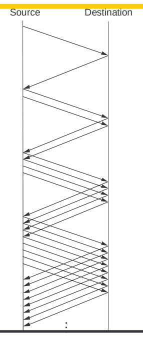
- How to adjust probe rate?
- Increase slowly while we believe there is bandwidth
  - Additive increase per RTT
  - Cwnd += 1 packet / RTT
- Decrease quickly when there is loss (went too far!)
  - Multiplicative decrease
  - Cwnd /= 2

TCP Sawtooth Pattern

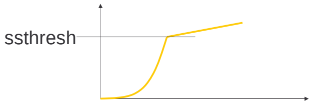


“Slow Start”

- Q: What is the ideal value of cwnd? How long will AIMD take to get there?
- Use a different strategy to get close to ideal value
  - Double cwnd every RTT
  - Cwnd *= 2 / RTT
  - Cwnd +=1 / packet received

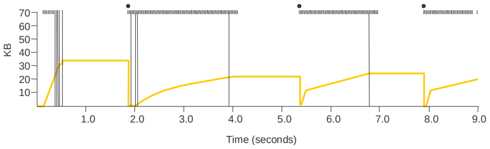


Combining Slow Start and AIMD



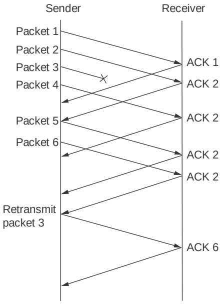
- Slow start is used whenever the connection is not running with packets: initially, and after timeouts
- But we don’t want to overshoot our ideal cwnd, so remember the last cwnd that worked with no loss
  - Ssthresh = cwnd after cwnd /= 2 on loss
  - Switch to AIMD once cwnd passes ssthresh

Example (Slow Start +AIMD)

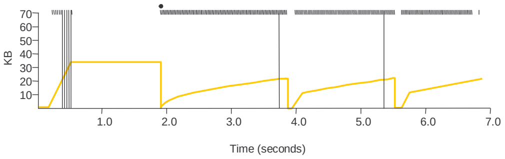


Fast Retransmit

- TCP uses cumulative acks, so duplicate acks start arriving after a packet is lost.
- We can use this fact to infer which packet was lost, instead of waiting for a timeout.
- 3 duplicate acks are used in practice



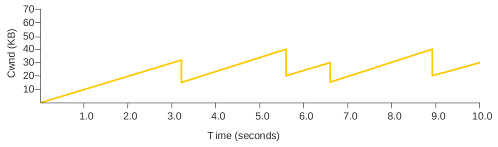
Example (with Fast Retransmit)



Fast Recovery

- After Fast Retransmit, use further duplicate acks to grow cwnd and clock out new packets, since these acks represent packets that have left the network.
- End result: Can achieve AIMD when there are single packet losses. Only slow start the first time.

Example (with Fast Recovery)



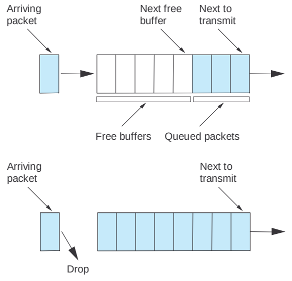
More Key Concepts

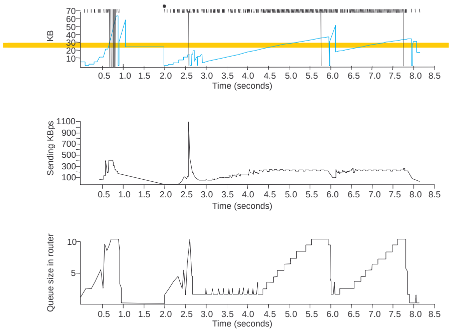
- TCP probes the network for bandwidth, assuming that loss signals congestion
- The congestion window is managed to be additive increase / multiplicative decrease
  - It took fast retransmit and fast recovery to get there
- Slow start is used to avoid lengthy initial delays
  - Ramp up to near target rate and then switch to AIMD

Why Congestion 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 (Yuck!)
  - 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

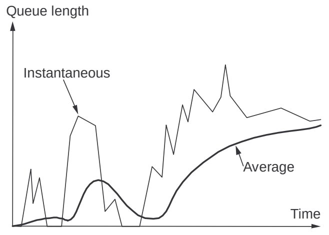
FIFO with Tail Drop





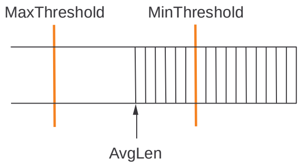
Incipient Congestion at a Router

- Sustained overload causes queue to build and overflow



Random Early Detection (RED)

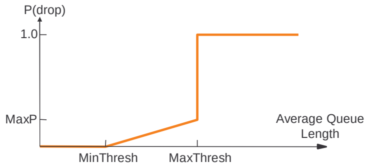
- Common approach is to have routers monitor average queue and send “early” signal to source when it builds by probabilistically dropping a packet



- Paradox: early loss can improve performance!

Red Drop Curve

- Start dropping a fraction of the traffic as queue builds
  - Expected drops proportional to bandwidth usage
  - When queue is too high, revert to drop tail
  - Nice theory, difficult to set parameters in practice



Explicit Congestion Notification (ECN)

- Why drop packets to signal congestion?
  - Drops are a robust signal, but there are other means …
  - We need to be careful though: no extra packets
- 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
- RED actually works by “marking” packets
  - Mark can be a drop or ECN signal if hosts understand ECN
  - Supports congestion avoidance without loss

Aside: TCP Vegas (Peterson ’94)

- RED needs router upgrades but no host upgrades
- Instead, can we upgrade host but not router?
- TCP Vegas looks at the difference between cwnd (the amount of outstanding data in the network) and that acknowledged from the other side in the last interval
  - Excess must be buffered in the network at router queues
  - Vegas slows down when it believes there is a queue and otherwise increases to use the available bandwidth

More Key Concepts

- We want to avoid congestion rather than control it after it has occurred
  - Think of in terms of the queues at routers
- Random early packet drops, rather than tail drop, can have unintuitive advantages
  - Signal congestion early, before we’re forced to drop repeatedly
- ECN signals congestion using bit in the IP header
  - No loss and no extra packets at overloaded times