CSE/EE 461 – Lecture 12

David Wetherall
djw@cs.washington.edu

Last Time

- More on the Transport Layer
- Focus
  - How do we <u>connect processes</u>?
- Topics
  - Naming processes
  - Connection setup / teardown
  - Flow control

Application
Presentation
Session
Transport
Network
Data Link
Physical

This Time

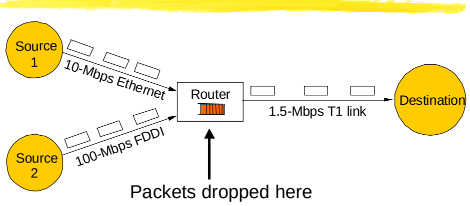
- Even more on the Transport Layer
- Focus
  - How do we <u>share bandwidth</u>?
- Topics
  - Congestion control
  - Fairness
  - Estimating round trip times (RTTs)

Application
Presentation
Session
Transport
Network
Data Link
Physical

Bandwidth Allocation

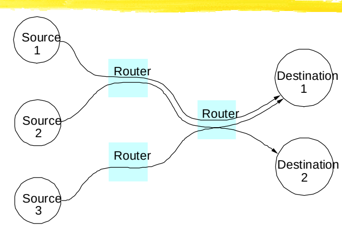
- How fast should the Web server send packets?
- Two big issues to solve!
- Congestion
  - sending too fast will cause packets to be lost in the network
- Fairness
  - different users should get their fair share of the bandwidth
- Often treated together (e.g. TCP) but needn't be

Cause of 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

Fairness



- Each <u>flow</u> from a source to a destination should get an equal share of the <u>bottleneck</u> 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" determine rate indirectly
- See Keshav 13.3 and 13.4 for more details.

Some Pros and Cons

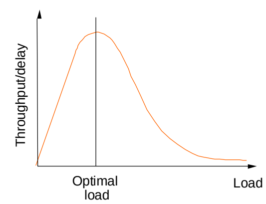
- Reservations don't work well with statistical multi-plexing unless you can characterize your traffic well.
- Adjusting based on network feedback leads to drops
- Network-based allocation needed to prevent cheating
- Host-based reduces implementation complexity
- Window schemes are more conservative than rate ones
  - They "stop" more quickly in the absence of ACKs

Design Choices

- TCP/Internet provides "best-effort" service
  - Network feedback, host controls via window.
  - No strong notions of fairness
- A different world in which there are QOS (quality of service) guarantees
  - Rate-based reservations natural choice for some apps
  - 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

- How do we compute the fairness of an allocation?
  - If all flows have an equal share at a router it's "fair"
  - But what if some flows don't want that much
  - How do we characterize how unfair unequal allocations are?
- Jain's fairness index:
  - For n flows each receiving a fraction f_i of the bandwidth
  - Fairness = $(\Sigma f_i)^2 / (n \times \Sigma f_i^2)$
  - Always between 0 and 1, 1 for equal allocations
  - If only k out of n flows get bandwidth, drops to k/n

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)
- Right interval is the round trip time (RTT) between sender and receiver
  - This varies greatly in the wide area (path length and queuing)
  - A good retransmission timer is important for good performance

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

Estimating RTTs (12.4.6 Keshav)

- 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
  - Set timeout to small multiple (2) of the estimate

Karn/Partridge Algorithm

- Problem:
  - RTT for retransmitted packets ambiguous
- Solution:
  - Don't measure RTT for retransmitted packets
  - Double retransmission timer on each subsequent timeout

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

Key Concepts

- Congestion
  - Queues build up and overflow inside network
  - TCP adapts sending rate based on network feedback
- Fairness
  - We want every flow to get it's fair share
  - Internet has very limited mechanisms for fairness
- Retransmission Timers
  - Important for good performance
  - Adapt based on recent samples (mean and variance)