P561: Network Systems
Week 6: Transport #2

Tom Anderson
Ratul Mahajan

TA: Colin Dixon

Administrivia

Fishnet Assignment #3
- Due Friday, 11/14, 5pm

Homework #3 (out soon)
- Due week 9 (11/24), start of class

2

Avoiding Small Packets

Nagle's algorithm (sender side):
- Only allow one outstanding segment smaller than the MSS
- A "self-clocking" algorithm
- But gets in the way for SSH etc. (TCP_NODELAY)

Delayed acknowledgements (receiver side)
- Wait to send ACK, hoping to piggyback on reverse stream
- But send one ACK per two data packets and use timeout on the delay
- Cuts down on overheads and allows coalescing
- Otherwise a nuisance, e.g, RTT estimation

Irony: how do Nagle and delayed ACKs interact?
- Consider a Web request

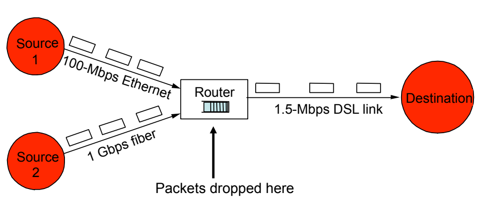
Bandwidth Allocation

How fast should a host, e.g., a web server, send packets?

Two considerations:
- 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

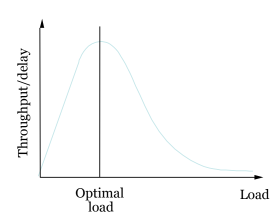


Buffer absorbs bursts when input rate > output
If sending rate is persistently > drain rate, queue builds
Dropped packets represent wasted work

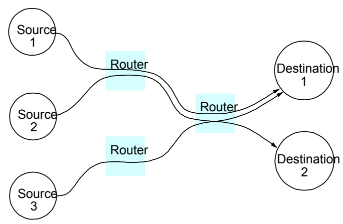
Evaluating Congestion Control

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
Even worse, can oscillate!



Fairness



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

Chapter 6, Figure 2

Evaluating Fairness

First, need to define what is a fair allocation.

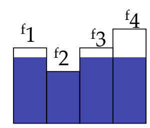
- Consider n flows, each wants a fraction f_i of the bandwidth

Min-max fairness:

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

Or proportional fairness

- Depends on path length …



Why is bandwidth allocation hard?

Given network and traffic, just work out fair share and tell the sources …

But:

- Demands come from many sources
- Needed information isn't in the right place
- Demands are changing rapidly over time
- Information is out-of-date by the time it's conveyed
- Network paths are changing over time

Designs affect Network services

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.

Case Study: TCP

The dominant means of bandwidth allocation today

Internet meltdowns in the late 80s ("congestion collapse") led to much of its mechanism

- Jacobson's slow-start, congestion avoidance [sic], fast retransmit and fast recovery.

Main constraint was zero network support and de facto backwards-compatible upgrades to the sender

- Infer packet loss and use it as a proxy for congestion

We will look at other models later …

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 Congestion Control

Goal: efficiently and fairly allocate network bandwidth
- Robust RTT estimation
- Additive increase/multiplicative decrease
  - oscillate around bottleneck capacity
- Slow start
  - quickly identify bottleneck capacity
- Fast retransmit
- Fast recovery

Tracking the Bottleneck Bandwidth

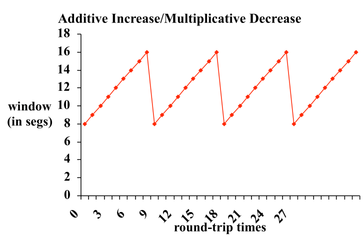
Sending rate = window size/RTT

Multiplicative decrease
- Timeout => dropped packet => sending too fast => cut window size in half
  - and therefore cut sending rate in half

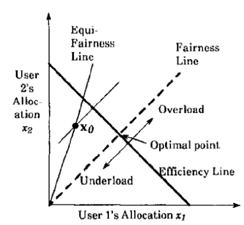
Additive increase
- Ack arrives => no drop => sending too slow => increase window size by one packet/window
  - and therefore increase sending rate a little

TCP "Sawtooth"

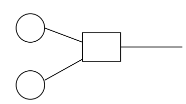
Oscillates around bottleneck bandwidth
- adjusts to changes in competing traffic



Why AIMD?



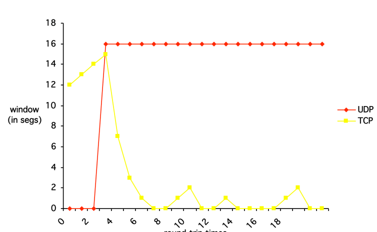
Two users competing for bandwidth:



Consider the sequence of moves from AIMD, AIAD, MIMD, MIAD.

What if TCP and UDP share link?

Independent of initial rates, UDP will get priority! TCP will take what's left.



What if two different TCP implementations share link?

If cut back more slowly after drops => will grab bigger share

If add more quickly after acks => will grab bigger share

Incentive to cause congestion collapse!
- Many TCP "accelerators"
- Easy to improve perf at expense of network

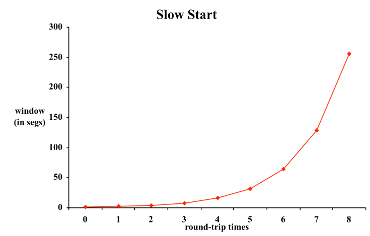
One solution: enforce good behavior at router

Slow start

How do we find bottleneck bandwidth?
- Start by sending a single packet
  - start slow to avoid overwhelming network
- Multiplicative increase until get packet loss
  - quickly find bottleneck
- Remember previous max window size
  - shift into linear increase/multiplicative decrease when get close to previous max ~ bottleneck rate
  - called "congestion avoidance"

Slow Start

Quickly find the bottleneck bandwidth



TCP Mechanics Illustrated

Source

Router

Dest

100 Mbps

0.9 ms latency

10 Mbps

0 latency

21

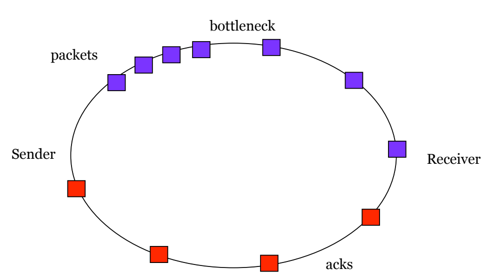
Slow Start vs. Delayed Acks

Recall that acks are delayed by 200ms to wait for application to provide data

But (!) TCP congestion control triggered by acks
- if receive half as many acks => window grows half as fast

Slow start with window = 1
- ack will be delayed, even though sender is waiting for ack to expand window

Avoiding burstiness: ack pacing

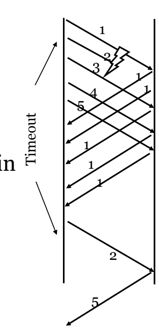


Window size = round trip delay * bit rate

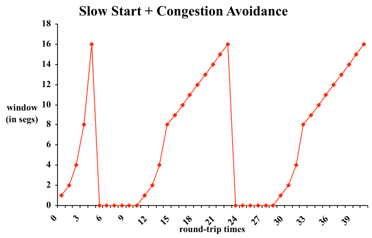
Ack Pacing After Timeout

Packet loss causes timeout, disrupts ack pacing
- slow start/additive increase are *designed* to cause packet loss

After loss, use slow start to regain ack pacing
- switch to linear increase at last successful rate
- "congestion avoidance"



Putting It All Together



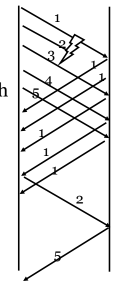
Timeouts dominate performance!

Fast Retransmit

Can we detect packet loss without a timeout?
- Receiver will reply to each packet with an ack for last byte received in order

Duplicate acks imply either
- packet reordering (route change)
- packet loss

TCP Tahoe
- resend if sender gets three duplicate acks, without waiting for timeout



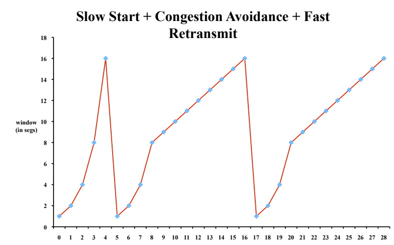
Fast Retransmit Caveats

Assumes in order packet delivery
- Recent proposal: measure rate of out of order delivery; dynamically adjust number of dup acks needed for retransmit

Doesn’t work with small windows (e.g. modems)
- what if window size <= 3

Doesn’t work if many packets are lost
- example: at peak of slow start, might lose many packets

Fast Retransmit

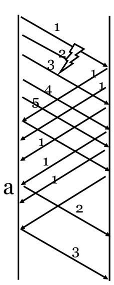


Regaining ack pacing limits performance

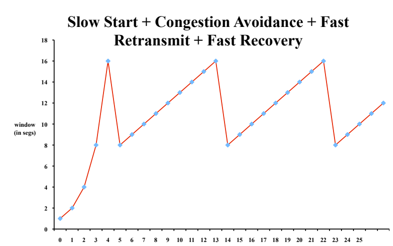
Fast Recovery

Use duplicate acks to maintain ack pacing
- duplicate ack => packet left network
- after loss, send packet after every other acknowledgement

Doesn’t work if lose many packets in a row
- fall back on timeout and slow start to reestablish ack pacing



Fast Recovery



TCP Performance (Steady State)

Bandwidth as a function of

- RTT?
- Loss rate?
- Packet size?
- Receive window?

TCP over 10Gbps Pipes

What's the problem?

How might we fix it?

TCP over Wireless

What's the problem?

How might we fix it?

What if TCP connection is short?

Slow start dominates performance

- What if network is unloaded?
- Burstiness causes extra drops

Packet losses unreliable indicator for short flows

- can lose connection setup packet
- Can get loss when connection near done
- Packet loss signal unrelated to sending rate

In limit, have to signal congestion (with a loss) on every connection

- 50% loss rate as increase # of connections

Example: 100KB transfer
100Mb/s Ethernet,100ms RTT, 1.5MB MSS

Ethernet ~ 100 Mb/s

64KB window, 100ms RTT ~ 6 Mb/s

slow start (delayed acks), no losses ~ 500 Kb/s

slow start, with 5% drop ~ 200 Kb/s

Steady state, 5% drop rate ~ 750 Kb/s

Improving Short Flow Performance

Start with a larger initial window

- RFC 3390: start with 3-4 packets

Persistent connections

- HTTP: reuse TCP connection for multiple objects on same page
- Share congestion state between connections on same host or across host

Skip slow start?

Ignore congestion signals?

Misbehaving TCP Receivers

On server side, little incentive to cheat TCP

- Mostly competing against other flows from same server

On client side, high incentive to induce server to send faster

- How?

37

Impact of Router Behavior on Congestion Control

Behavior of routers can have a large impact on the efficiency/fairness of congestion control

- buffer size
- queueing discipline (FIFO, round robin, priorities)
- drop policy -- Random Early Drop (RED)
- Early congestion notification (ECN)
- Weighted fair queueing
- Explicit rate control

Note that most solutions break layering

- change router to be aware of end to end transport

TCP Synchronization

Assumption for TCP equilibrium proof is that routers drop fairly

What if router's buffers are always full?

- anyone trying to send will experience drop
  - timeout and retry at reduced rate
- when router sends a packet, triggers an ack
  - causes that host to send another packet, refill buffers, causes other hosts to experience losses

One host can capture all of the bandwidth, even using TCP!

Router Buffer Space

What is the effect of router queue size on network performance?

- What if there were infinite buffers at each router?
  - what would happen to end to end latency?
- What if only one packet could be buffered?
  - what would happen if multiple nodes wanted to share a link?

Subtle interactions between TCP feedback loop and router configuration

- rule of thumb: buffer space at each router should be equal to the end to end bandwidth delay product (how?)

Congestion Avoidance

TCP causes congestion as it probes for the available bandwidth and then recovers from it after the fact

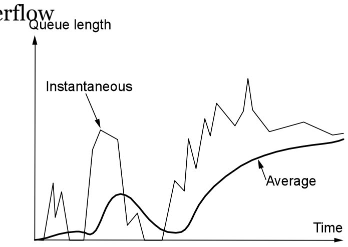
- Leads to loss, delay and bandwidth fluctuations (Yuck!)
- We want congestion avoidance, not congestion control

Congestion avoidance mechanisms

- Aim to detect incipient congestion, before loss. So monitor queues to see that they absorb bursts, but not build steadily

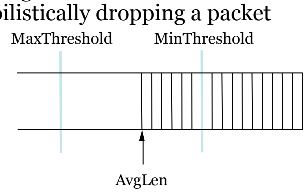
Incipient Congestion at a Router

Sustained overload causes queue to build and overflow



Random Early Detection (RED)

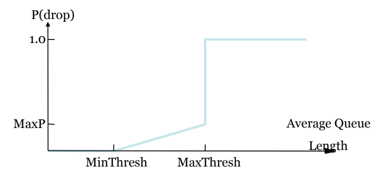
Have routers monitor average queue and send "early" signal to source when it builds by probabilistically dropping a packet



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



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

Difficulties with RED

Nice in theory, hasn't caught on in practice.

Parameter issue:
- What should dropping probability (and average interval) be?
- Consider the cases of one large flow vs N very small flows

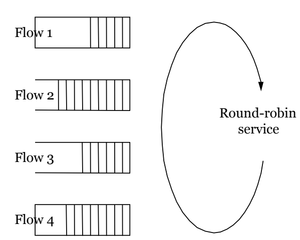
Incentive issue:
- Why should ISPs bother to upgrade?
  - RED doesn't increase utilization, the basis of charging
- Why should end-hosts bother to upgrade?
  - The network doesn't support RED

Fair Queuing (FQ)

FIFO is not guaranteed (or likely) to be fair
- Flows jostle each other and hosts must play by the rules
- Routers don't discriminate traffic from different sources

Fair Queuing is an alternative scheduling algorithm
- Maintain one queue per traffic source (flow) and send packets from each queue in turn
  - Actually, not quite, since packets are different sizes
- Provides each flow with its "fair share" of the bandwidth

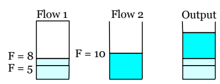
Fair Queuing



Fair Queuing

Want to share bandwidth
- At the "bit" level, but in reality must send whole packets

Approximate using <u>finish</u> times for each packet
- Let A be a virtual clock that is incremented each time all waiting flows send one bit of data
- For a packet i in a flow: Finish(i) = max(A, F(i-1)) + length-in-bits
- Send in order of finish times, without pre-emption



More generally, assign <u>weights</u> to queues (Weighted FQ, WFQ)
- how to set them is a matter of policy

Implementing WFQ

Sending in order of F(i) requires a priority-queue
- O(log(n)) work per packet

Tracking F(i)s requires state for each recently active flow
- May be a large number of flows at high speeds

Q: Can we approximate with less work/state?

Deficit Round Robin
- For each round, give each flow a quantum of credit (e.g., 500 bits), send packets while credit permits, and remember leftover for next round
- Very close to WFQ

Stochastic Fair Queuing
- Hash flows into a fixed number of bins
- Fair except due to collisions

WFQ implication

What should the endpoint do, if it knows router is using WFQ?

51

Traffic shaping

At enterprise edge, shape traffic:
- Avoid packet loss
- Maximize bandwidth utilization
- Prioritize traffic
- No changes to endpoints (as with NATs)

Mechanism?

52

TCP Known to be Suboptimal

Small to moderate sized connections

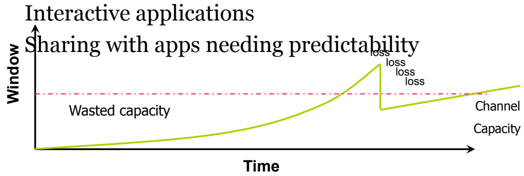
Paths with low to moderate utilization

Wireless transmission loss

High bandwidth; high delay

Interactive applications

Sharing with apps needing predictability



Observation

Trivial to be optimal with help from the network; e.g., ATM rate control
- Hosts send bandwidth request into network
- Network replies with safe rate (min across links in path)

Non-trivial to change the network

Question

Can endpoint congestion control be near optimal with *no* change to the network?

Assume: cooperating endpoints
- For isolation, implement fair queueing
- PCP does well both with and without fair queueing

PCP approach: directly emulate optimal router behavior!

Congestion Control Approaches

	Endpoint	Router Support
Try target rate for full RTT; if too fast, backoff	TCP, Vegas, RAP, FastTCP, Scalable TCP, HighSpeed TCP	DecBit, ECN, RED, AQM
Request rate from network; send at that rate	**PCP**	ATM, XCP, WFQ, RCP

PCP Goals

1. Minimize transfer time
2. Negligible packet loss, low queueing
3. Work conserving
4. Stability under extreme load
5. Eventual fairness

TCP achieves only the last three (with FIFO queues)

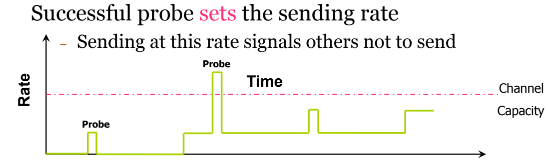
PCP achieves all five (in the common case)

Probe Control Protocol (PCP)

Probe for bandwidth using short burst of packets
- If bw available, send at the desired uniform rate (paced)
- If not, try again at a slower rate

Probe is a request

Successful probe sets the sending rate
- Sending at this rate signals others not to send



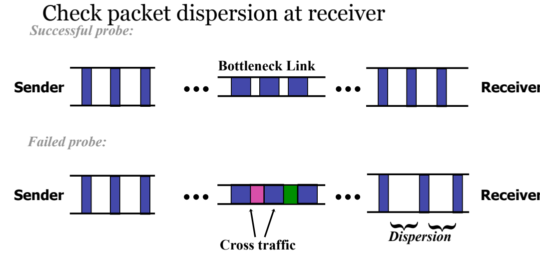
PCP Mechanisms

Mechanism	Description	Goal
Probe followed by direct jump	Send short bursts to check for available bandwidth; if successful, send at that rate	low loss, min response time
probabilistic accept	Accept probes taking into account noise.	min response time, fairness
rate compensation	Drain queues, detect cross traffic, correct errors.	low loss, low queues
periodic probes	Issue probes periodically to check for available bandwidth.	work conserving
binary search	Use binary search to allocate the available bandwidth.	min response time, work conserving
exponential backoff	Adjust probe frequency to avoid collision.	Stability
history	Use heuristics to choose initial probe rate.	min response time
tit-for-tat	Reduce speed of rate compensation.	TCP compatibility

Probes

Send packet train spaced to mimic desired rate

Check packet dispersion at receiver

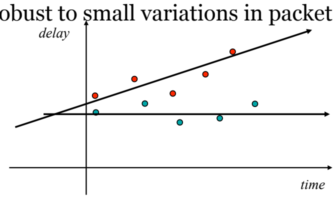


Probabilistic Accept

Randomly generate a slope consistent with the observed data
- same mean, variance as least squares fit

Accept if slope is not positive

Robust to small variations in packet scheduling



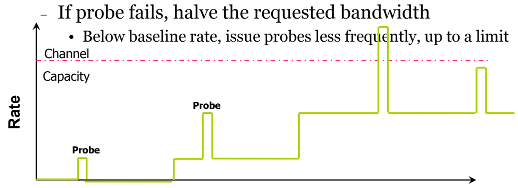
Rate Compensation

Queues can still increase:
- Failed probes, even if short, can result in additional queueing
- Simultaneous probes could allocate the same bandwidth
- Probabilistic accept may decide probe was successful, without sufficient underlying available bandwidth

PCP solution
- Detect increasing queues by measuring packet latency and inter-packet delay
- Each sender decreases their rate proportionately, to eliminate queues within a single round trip
- Emulates AIMD, and thus provides eventual fairness

Binary Search

Base protocol: binary search for channel capacity
- Start with a baseline rate: One MSS packet per round-trip
- If probe succeeds, double the requested bandwidth
- If probe fails, halve the requested bandwidth
  - Below baseline rate, issue probes less frequently, up to a limit



History

Haven't we just reinvented TCP slow start?
- Still uses *O(log n)* steps to determine the bandwidth
- Does prevent losses, keeps queues small

Host keeps track of previous rate for each path
- Because probes are short, ok to probe using this history
- Currently: first try 1/3rd of previous rate
  - If prediction is inaccurate/accurate, we halve/double the initial probe rate

TCP Compatibility

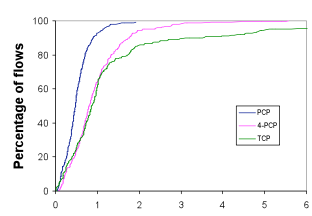
TCP increases its rate regardless of queue size
- Should PCP keep reducing its rate to compensate?

Solution: PCP becomes more aggressive in presence of non-responsive flows
- If rate compensation is ineffective, reduce speed of rate compensation: "tit for tat"
- When queues drain, revert to normal rate compensation

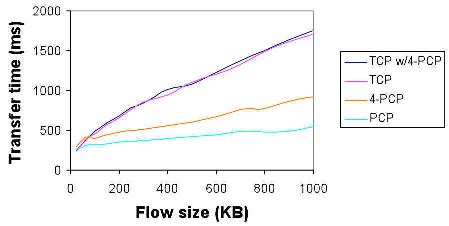
Otherwise compatible at protocol level
- PCP sender (receiver) induces TCP receiver (sender) to use PCP

Performance

User-level implementation
- 250KB transfers between every pair of US RON nodes
- PCP vs. TCP vs. four concurrent PCP transmissions



Is PCP Cheating?



Related Work

Short circuit TCP's slow-start: TCP Swift Start, Fast Start

Rate pacing: TCP Vegas, FastTCP, RAP

History: TCP Fast Start, MIT Congestion Manager

Delay-based congestion control: TCP Vegas, FastTCP

Available bandwidth: Pathload, Pathneck, IGI, Spruce

Separate efficiency & fairness: XCP

Roadmap – Various Mechanisms

Simple to build,

Weak assurances

Classic Best Effort	FIFO with Drop Tail
Congestion Avoidance	FIFO with RED
Per Flow Fairness	Weighted Fair Queuing
Aggregate Guarantees	Differentiated Services
Per Flow Guarantees	Integrated Services

Complex to build,

Strong assurances

Lead-in to Quality of Service

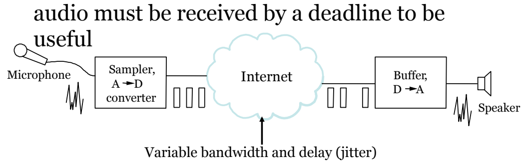
Our network model so far is "Best Effort" service
- IP at routers: a shared, first come first serve (drop tail) queue
- TCP at hosts: probes for available bandwidth, causing loss

The mechanisms at routers and hosts determine the kind of service applications will receive from the network
- TCP causes loss and variable delay, and Internet bandwidth varies!

Q: What kinds of service do different applications need?
- The Web is built on top of just the "best-effort" service
- Want better mechanisms to support demanding applications
- Once we know their needs we'll revisit network design …

VoIP: A real-time audio example

VoIP is a real-time service in the sense that the audio must be received by a deadline to be useful



Real-time apps need assurances from the network

Q: What assurances does VoIP require?

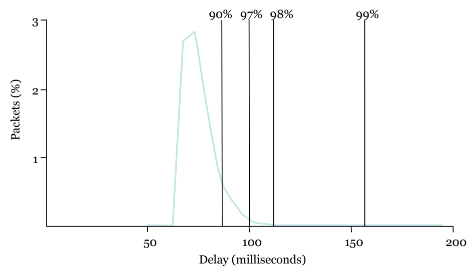
Network Support for VoIP

Bandwidth
- There must be enough on average
- But we can tolerate to short term fluctuations

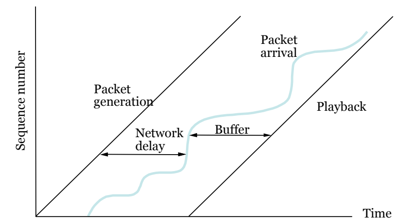
Delay
- Ideally it would be fixed
- But we can tolerate some variation (jitter)

Loss
- Ideally there would be none
- But we can tolerate some losses. (How?)

Example: Delay and Jitter

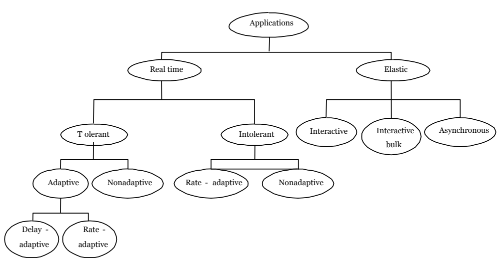


Tolerating Jitter with Buffering



Buffer before playout so that most late samples will have arrived

Taxonomy of Applications



Specifying Application Needs

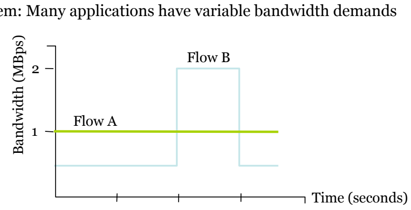
First: many applications are elastic, and many real-time applications are tolerant of some loss/delay and can adapt to what the network can offer

Second: we need to a compact descriptor for the network
- Analogous to SLA

Delay: can give a bound for some percentile
Loss: can give a bound over some period
What about bandwidth? Many applications are bursty ...

Specifying Bandwidth Needs

Problem: Many applications have variable bandwidth demands

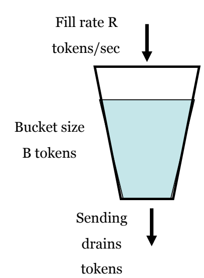


Same average, but very different needs over time. One number. So how do we describe bandwidth to the network?

Token Buckets

Common, simple descriptor

Use tokens to send bits
Average bandwidth is R bps
Maximum burst is B bits



Supporting QOS Guarantees

1. Flowspecs. Formulate application needs
   - Need descriptor, e.g. token bucket, to ask for guarantee
2. Admission Control. Decide whether to support a new guarantee
   - Network must be able to control load to provide guarantees
3. Signaling. Reserve network resources at routers
   - Analogous to connection setup/teardown, but at routers
4. Packet Scheduling. Use different scheduling and drop mechanisms to implement the guarantees
   - e.g., set up a new queue and weight with WFQ at routers

The need for admission control

Suppose we have an <r,b> token bucket flow and we are interested in how much bandwidth the flow receives from the network.

Consider a network with FIFO nodes. What rate does the flow get?

Now consider a network with (W)FQ nodes. What rate does the flow get?

Now consider a network with (W)FQ nodes where $w(i) = r(i)$ and $\sum w(i) = W <$ capacity at each node. What rate does the flow get?

Bounding Bandwidth and Delay

WFQ with admission control can bound bandwidth and delay. Wow! (Parekh and Gallagher GPS result)

For a single node:
- Bandwidth determined by weights: g(i) = C * w(i)/W
- E2E delay <= propagation + burst/g(i) + packet/g(i) + packet/C

For multiple nodes:
- Bandwidth is determined by the minimum g(i) along the path
- E2E delay pays for burst smoothing only once, plus further transmission and pre-emption delays

GPS Example

Assume connection has leaky bucket parameters (16KB, 150Kbps), and crosses 10 hops, all link bandwidths are 45Mb/s, and the largest packet size is 8KB.

What g will guarantee an end-to-end delay of 100ms, assuming total propagation delay of 30ms?

From before:
- E2E delay <= prop + burst/g(i) + N* packet/g(i) + N*packet/C
- 0.1 <= 0.03 + (16K*8)/g+10*8K*8/g+10*8K*8/45*10^6
- Solving, we have a g of roughly 13 Mbps

Moral: may need to assign high rates to guarantee that worst case burst will have acceptable E2E delay

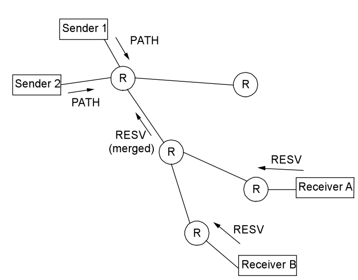
IETF Integrated Services

Fine-grained (per flow) guarantees
- Guaranteed service (bandwidth and bounded delay)
- Controlled load (bandwidth but variable delay)

RSVP used to reserve resources at routers
- Receiver-based signaling that handles failures

WFQ used to implement guarantees
- Router classifies packets into a flow as they arrive
- Packets are scheduled using the flow's resources

Resource Reservation Protocol (RSVP)



RSVP Issues

RSVP is receiver-based to support multicast apps

Only want to reserve resources at a router if they are sufficient along the entire path

What if there are link failures and the route changes?

What if there are sender/receiver failures?

IETF Differentiated Services

A more coarse-grained approach to QOS

- Packets are marked as belonging to a small set of services, e.g, premium or best-effort, using the TOS bits in the IP header

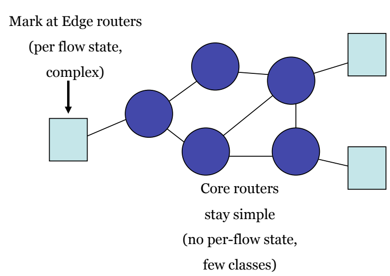
This marking is policed at administrative boundaries

- Your ISP marks 10Mbps (say) of your traffic as premium depending on your service level agreement (SLAs)
- SLAs change infrequently; much less dynamic than Intserv

Routers understand only the different service classes

- Might separate classes with WFQ, but not separate flows

Two-Tiered Architecture



DiffServ Issues

How do ISPs provision?

- Traffic on your access link may follow different paths inside ISP network. Can we provide an access link guarantee efficiently?

What's the policy?

- Which traffic is gold, which silver, etc.?

Overprovisioning, other issues

An alternative:

- Provide more capacity than load; it's all a cost tradeoff
- Bandwidth to user limited mainly by their access capacity
- Delay through network limited mainly by propagation delay

Deploying QOS:

- What good is it if only one ISP deploys?
- Incentives for single ISP for distributed company using VoIP
- And incentive for inter-provider agreements
- Network QOS as an extension of single box packet shapers