6.033 Lecture 12 3/15/09
Congestion Control Wrapup and Internet Routing Intro

Where are we:

Networks
Saw 3 layer design

E2E <-- finish
Net <-- discuss internet routing
Link

Been talking about E2E layer -- in particular, how to implement reliability.

Saw ack protocol, to provide exactly once.

Problem was that waiting after every send for the ack gave low throughput.

So we developed a window-based protocol to keep the network channel busy.

Set our windows to rate x RTT packets big

Estimating RTT is easy, but estimating rate is hard, because we don't know what the slowest point in the network is.

If we overestimate rate, we have a problem, because we get congestion collapse. (Show slide)

Really don't want this to happen in the core of the internet as it will break service for everyone.

So we introduced congestion control, designed to avoid overshooting rate.

Two key ideas:

- Increase timeouts, using exponential backoff

- Decrease window size. (WS 1 == wait for ack before sending next packet), using additive increase, multiplicative decrease (also exponential backoff)

Show protocol slide.

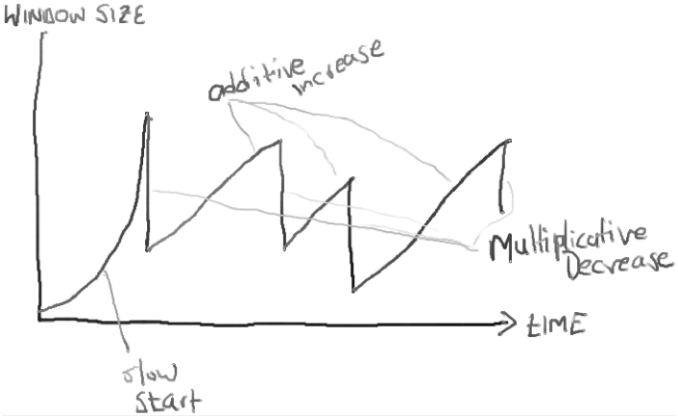
Problem: can take a long time for additive increase to get to big window size, so use "slow start".

Slow start -- while connection is starting only

On each ack:

CW = CW * 2 (max receiver window)

Show diagram:



Note that receiver can also shrink the receive window, which will cause sender to backoff if it is overloaded.

Why does TCP work?

High order bit -- "ack clocking" : there's some bottleneck, with rate R. Packets sent from that bottleneck at rate R, arrive at receiver at rate R, and acks back at rate R. Next packet isn't sent until previous packet is ack'd.

All this other stuff -- backing off aggressively, etc, is really about helping us deal with dynamics (e.g., sudden changes in bottleneck rate), and avoiding overwhelming the bottleneck link.

Is TCP "fair"?

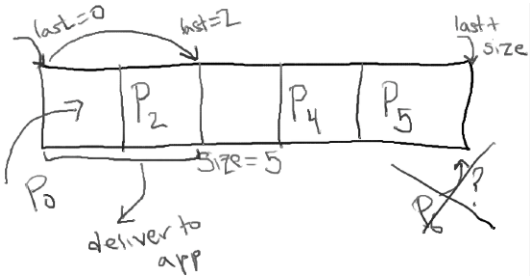
Note that applications aren't required to use TCP in which case they won't obey congestion rules. Applications can also use multiple TCP connections.

So why does this work? Apparently because most traffic is TCP.

Ok -- so now we know how to provide exactly once delivery that does a good job of keeping network busy without causing congestion. What's left?

Reliability requires us to reorder packets -- pretty easy using a reorder buffer.

Reordering -- packets may arrive at the receiver out of order (show slide)




```
recv(p)
        slot = p.sno - last
        if (slot > last + size)
                drop
        else
                new = slot is empty
                if (new) put p in slot
                ack p
        if (slot == last and new)
                deliver prefix to app
                slot += size(prefix)
```

Just to recap, let's look at the packets of IP and TCP and see how all these features are actually provided.

Now, pop the stack and return to the question of how routes get built in Internet. Remember, each node has a *forwarding table* which tells it how to reach a given destination.

How to compute forwarding table? Manually -- not scalable.

Centrally -- not a good idea (why?)

- need a routing algorithm to collect
- collection requires many messages
- hard to adapt to changes

Path Vector Algorithm -- Distributed

Each node maintains a forwarding table T , with:

Dest Link Path

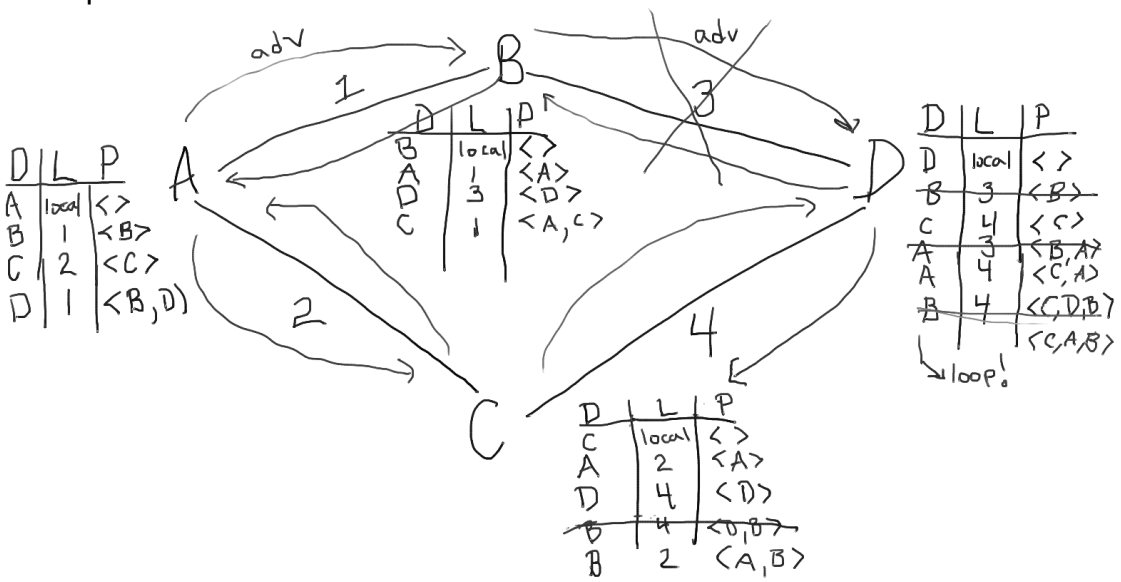
Two steps:

advertise (periodic -- e.g. every n seconds)
send T to neighbors

integrate(N,neigh, link) -- table N from neighbor neigh on link
merge(T,N,neigh,link)

Merge(T,N,neigh,link)
for each dest d w/ path r in N:
if d not in T, add (d, link, neigh ++ r) to T
if d is in T, replace if (neigh ++ r) is shorter than old path

Example:



(If everybody picks best path to every dest, you can see that for a network with most distant nodes separated by N hops, in N rounds everyone will know how to reach everyone else in N steps.)

Problems:

- failures / changes -- repeat advertisements periodically,
  remove paths in your table that aren't re-advertised
  (e.g., a path P that begins with router R should be in the
  next advertisement from R.)

- permanent loops?
  - won't arise if we add a rule that we don't pick paths with ourselves
  in them; this is what we need the path for!

- temporary loops -- arise because two nodes may be slightly out of date
  example
  - soln: add send count -- "TTL" -- to packet

- graph changes -- same as failures

How does this work on the Internet:

At first, internet was a small network like this

What is the problem with using path vector here?

Network is huge

> 1 B nodes on network

Each router needs to know how to reach of these billions of computers

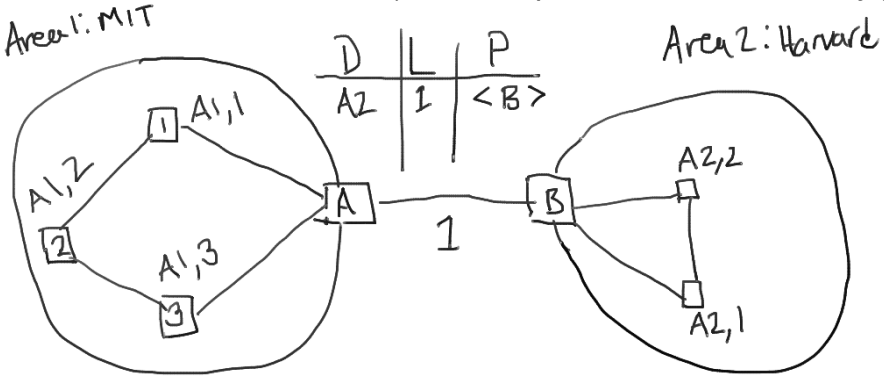
With pure path vector, each node has a multi-billion entry table (requiring
gigabytes of storage)

Each router has to send these gigabyte tables to each of its neighbors;
millions of advertisements propagating around. Disaster.

Solution: hierarchical routing

Subdivide net into areas; with multiple levels of routing

One node representative of each area; perform path vector at area level. Within each area, free to do whatever. (For example, use more hierarchy.)



Protocol that runs this is on Internet is BGP; will learn about next time in recitation. Destinations on the internet (e.g., A2) are actually IP address prefixes that the routers can route to.

D	L	P
173.203.*.*	1	<B>

Good -- tables are much smaller, fewer advertisements as internal changes aren't advertised.

Bad -- for this be scalable, contiguous ranges of addresses have to be contained within areas, which means that ranges of addresses are owned by particular regions.

How are IP addresses assigned?

There is an organization called IANA, that allocates IP addresses to region (per continent) bodies. In NA, that org is called ARIN. To request IPs from ARIN, you have to prove that you are an ISP of a certain minimum size (e.g., a /20, which means you will get a block of addresses all with the same first 20 bits, or 2^12 = 4096 addresses.) You have to prove you are using these efficiently. Not very expensive (a few thousand per year for large network.s)