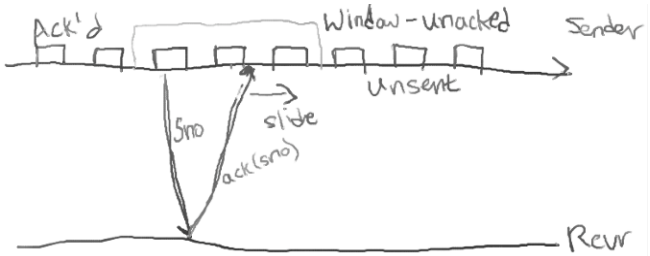
6.033 Lecture 13 **3/18/09**

Sam Madden

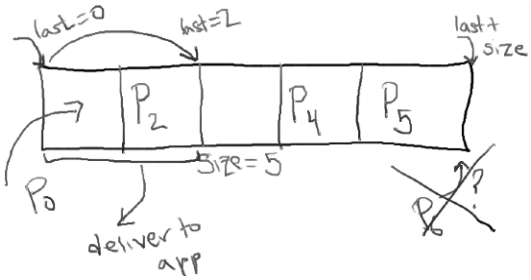
Congestion Control

Last time, we saw how the end to end layer ensures exactly once delivery, using windows to efficiently make use of the network.



Couple of loose ends:

- Reordering -- packets may arrive at the receiver out of order




```
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)
```

Cumulative acks

So far, have presented acks as being *selective* -- for a specific packet

In Internet, sometimes *cumulative acks* are used

in figure above, when p1 arrives, would send ack(2), indicating that all packets up to 2 have been received.

+ insensitive to lost acks
    (if using selective acks, a dropped ack requires a packet retransmit. Here, if the ack for P2 was lost, when P0 arrives, receiver knows P2 arrived.)

- sender doesn't know what was lost

    here, sender has sent p3, p4, and p5; p4 and p5 have been receiver, but only receives an ack for up to p2.

    can lead to excessive retransmission

In the TCP, both cumulative and selective acks are used:

    common case, use cumulative, so that a lost ack isn't problematic
    when acking a packet for which later packets have arrived,
        use selective acks, so that sender knows what has arrived

Choosing the window size

Last time, we send that the window size should be:

    Window = Rate x RTT

Where Rate is the maximum rate the receiver can accept packets at

This will keep the receiver busy all the time

E.g., suppose receiver can handle 1 packet / ms , and RTT is
    10 ms; then window size needs to be 10 packets for receiver to always be busy

2 questions:

1) How does the receiver know how fast it can process packets?
2) What if the network can't deliver packets as fast as the receiver can process?

1) It doesn't. In practice it doesn't matter that much -- just set it to be big enough. For example, on 100 Mbit ethernet, with 10 ms RTT,

window = 100 Mbit/sec * .01 sec = 1 Mbit = 128 KB

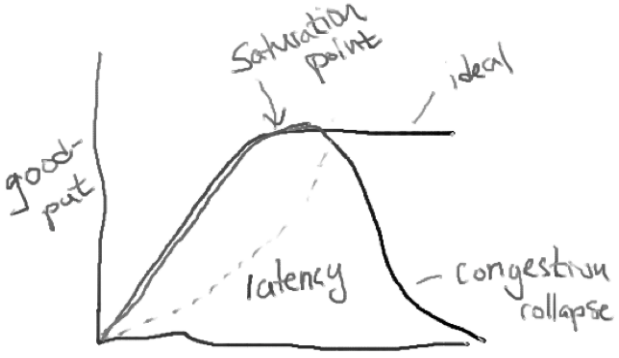
So what if the network can't handle packets at this rate -- is that a problem?

Suppose sender and receiver can both communicate at 3 pkts / sec, but that there is a bottleneck router somewhere between the sender and receiver that can only send 1 pkts / sec. Suppose RTT is 2 sec (so receive window is 6 pkts)

After 1 sec, there will be 2 packets in this router's queues
After 2 seconds, there will be 4 packets

Congestion collapse = delivered load constantly exceeds available bandwidth

diagram:



Notice that if we could adapt RTT on sender fast enough, that would help, since we'd sender fewer duplicates.

But we could still get into trouble if we have a big window and there is a slow bottleneck.

Why does congestion arise?

Routers in the middle of the Internet are typically big and fast -- much faster than the end hosts. So the scenario above shouldn't arise, should it?

Issue: sharing. Sources aren't aware of each other, and so may all attempt to use some link at the same time (e.g., flash crowds). This can overwhelm even the beefiest routers.

Example: 9/11 -- everyone goes to CNN. Apple -- releases new products.

What do do about it:

Avoid congestion:

Increase resources? Expensive and slow to react; not clear it will help

Admission control? Used in telephone network. Some web servers do this, to reduce load. But its hard to do this inside of the network, since routers don't really have a good model of applications or the load they may be presenting.

Congestion control -- ask the senders to slow down.

How do senders learn about congestion?

Options:

1) Router sends feedback (either directly to the sender, or by flagging packets, which receiver then notices and propagates in its acks.)

Works, but can be problematic if network is congested because these messages may not get through (or may be very delayed)

2) Assume that packet timeouts indicate congestion.

What to do about congestion:

- Increase timeouts.
- Decrease window size.

Need to do this very aggressively -- otherwise it is easy for congestion to collapse to arise. CC is hard to recover from.

TCP does both of these:

Timeouts: exponential backoff

On retransmitted packet, set:

set timeout = timeout * c ; c = 2

up to some maximum

timeout increases exponentially

After ack's start arriving (losses stop)
, timeout is reset based on RTT EWMA (as in last class)

2 windows: receive window + "Congestion Window":

Window size = min(congestion window, receiver windoow)

On retransmitted packet:

CW = CW/2

(Min size = 1)

On ack'd packet:

CW = CW + 1 (up to receive window size)

To initialize window quickly, use TCP "slow start":

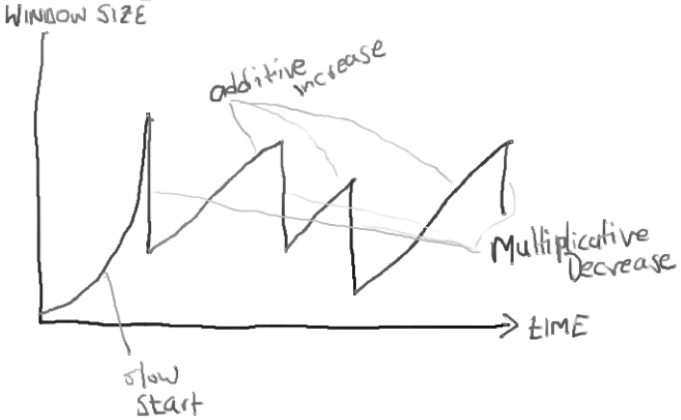
At the start of the connection, double CW up to receive window, unless losses occur:

Slow start -- while connection is starting only

On each ack:

CW = CW * 2 (max receive window)

Show diagram:



Is TCP "fair"?
no ; 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.