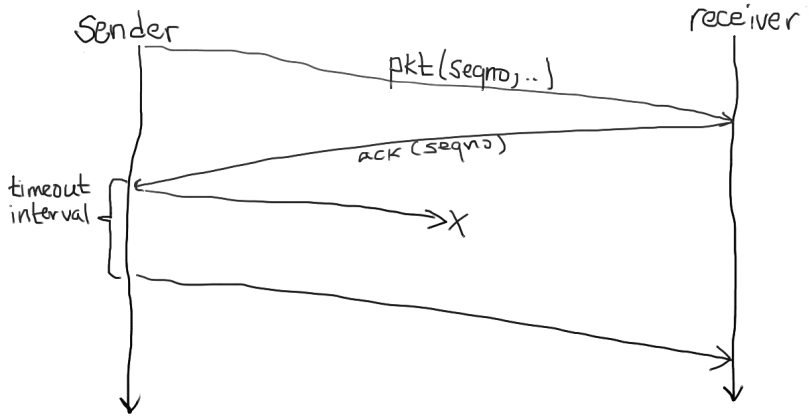
6.033 Lecture 11 -- Network Layering and Reliability 3/10/2010

Last lecture -- 3 layer networks design; started talking about reliability, saw **at least once delivery**



How to set the timer interval?

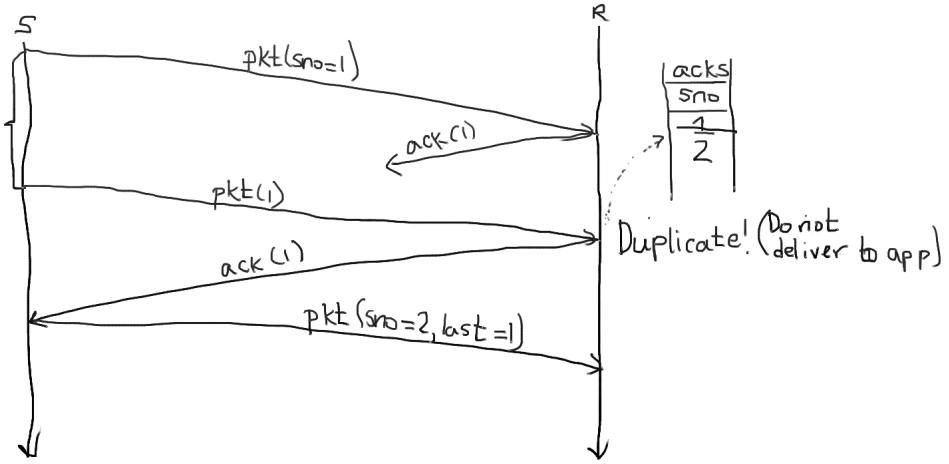
Measure round trip time (RTT) and adjust.
(show slides)

Ok, so now we set the timer appropriately. Ensures that at least one copy of every data item arrives.

Can we have duplicates? (yes, why?)
What to do about this?

At most once delivery

On receiver, keep track of which packets have been ack'd (in a list). When a duplicate packet arrives, resend ack, but don't deliver to app.
Diagram:



When can you remove something from list of acks? Since acks can be delayed or lost, could receive resent packets quite a bit later. One typical solution is to attach last msg id received on messages from sender.

At least once delivery and at most once delivery together are **exactly once delivery**.

Bit of a misnomer.

Issues:

- delivery may not be possible
- Suppose receiver crashes after receiving message but before sending ack; reboots --> did it process the message or not, what if it receives message again?

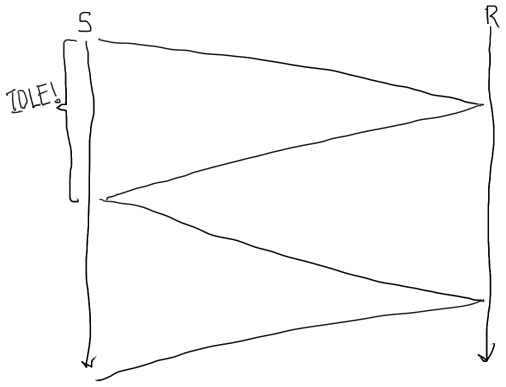
We will talk much more about building reliable systems and these issues after spring break.

- What does an ack mean? Typically just that the E2E layer handed the packet off to the application, *not that the application processed the message.*

Note that the protocol thus far deliver stuff in order b/c it only has one outstanding message at a time.

At this point, we've built a streaming, exactly once, in order delivery mechanism. Problem is it is SLOOW.

Show diagram:



Suppose RTT is 10 ms. Can only send 100 packets / second. If packets are 1000 bits, then we are only sending at 100 Kbit/sec. Not good!

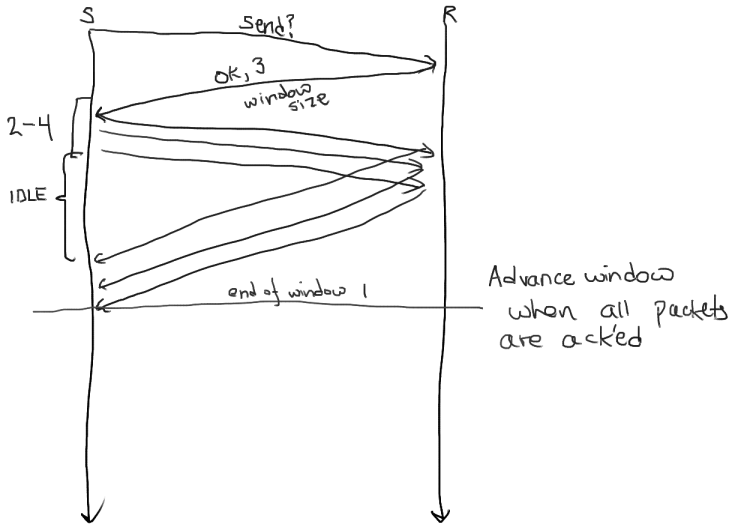
Can we just make packets bigger?

How big would they have to be for gig-e? 10,000 times bigger --> 10 mbits. Huge! Retransmissions are super expensive, and we aren't doing a very good job of multiplexing the network.

Solution: smaller packets, but multiple outstanding at one time.

"window" of outstanding packets

Show diagram:



Still wasteful, since sender waits an RTT. Solution: slide window -- when ack arrives, advance window by one.

(show slides)

Note that if the window size is too small, may still end up waiting.

So, how big to make windows? Want to be able to "cover" delay.
Suppose we have a measure of RTT. (delay)

And suppose we know the maximum rate the network can send (either because of links, delays in the network, or limits at the sender or receiver), (bandwidth)

Max packets outstanding is then

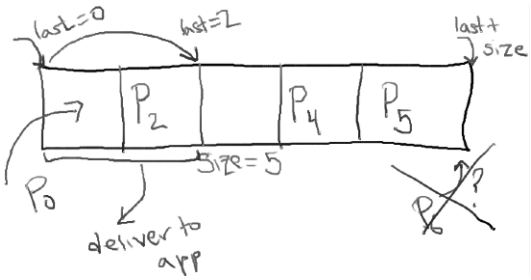
window = RTT (delay) x rate (bw)

"bandwidth delay product"

Receiver continuously sends window size in packets; can reduce size if overloaded

Of course, this assumes we can measure the rate accurately, which turns out to be hard because rate depends on what is happening inside the network (e.g., what other nodes everywhere else inside the network is doing.) Come back to this.

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)
```

How to measure rate at which receiver / network can handle packets, and back off as needed? Congestion control.

Choosing the window size

As we saw, window size should be:

Window = Rate x RTT

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 toalways 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

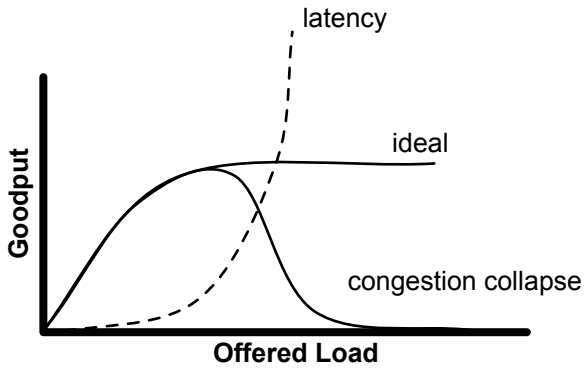
So what if the network can't handle packets at this rate -- is that a problem?

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)

(show slides) 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, so may all attempt to use some link at the same time (e.g., flash crowds). 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. (WS 1 == wait for ack before sending next packet)

Need to do this very aggressively -- otherwise it is easy for congestion to collapse to arise. CC is hard to recover from. Both slow down sending (why?)

<u>TCP does both of these:</u>

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)

Window: throttle the window based on congestion

sender window = max(congestion window, receiver window)

On retransmitted packet:

CW = CW/2

(Min size = 1)

On ack'd packet:

CW = CW + 1

How to set window size initially? Start at 1, then to initialize window quickly, use TCP "slow start":

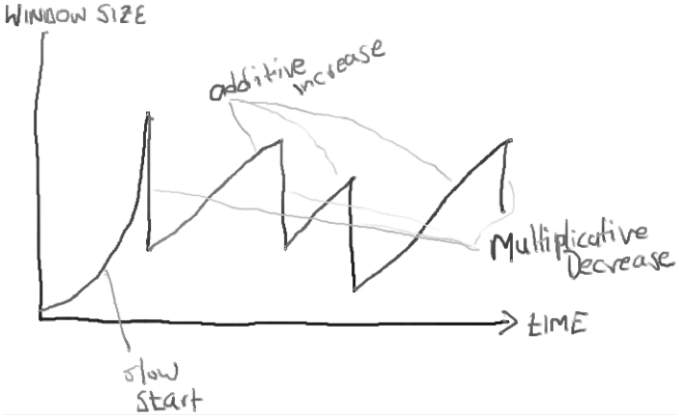
At the start of the connection, double CW up to receiver window, unless losses occur:

Slow start -- while connection is starting only

On each ack:

CW = CW * 2 (max receiver 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.