6.033 Lecture 12 3/16/09

Sam Madden

End to End Layer

Last time: network layer -- how to deliver a packet across a network of multiple links

Recall that network layer is **best effort**, meaning:

- packets may be lost, reordered, garbled
- losses due to no route, queue overflow, packet corruption / collisions, etc.
- reordering due to packets taking different routes, retransmissions

Network layer is also message oriented.

E2E layer addresses these limitations. Some things applications might want:

1. Connection, stream of data, rather than having to explicitly divide data into messages

2. All packets to arrive (reliable)

3. No duplicates (at most once)

4. Packets arrive in order

5. Efficiency

Not all applications need all of these -- for example, in voice chat app, you might prefer more predictable latency w/ some packets being dropped, rather than all packets arriving with unpredictable latency.

This list above is roughly what Internet's TCP gives you. Internet provides other protocols, like UDP (none of the above) or RTP (streaming, but not reliable)

Different E2E layers will provide different subsets of these features. Goal is to build these features without modifying network layer.

E2E Interface

Diagram: (including port->app table), packets

Streaming connection (add entries to port->app table)

src	dest
conn = open_stream(dest, port)	conn = listen(port)
send(conn, bytes1)	
send(conn, bytes2)	bytes = recv(conn)
close_stream(con)	

E2E layer fragments streams up into packets that it sends to network layer.

Reliability --> At least once

All packets arrive, and no packets are corrupted

For corruption, typically packets have some kind of checksum, which is checked when the packet is received.

How to ensure all packets arrive?

Idea:

have receive send an acknowledgement for every packet. Set a timer on the sender, and when the ack doesn't arrive after that much time, resend.

associate a unique sequence number with each packet.

attach sequence number to the ack.

Diagram:

How to set the timer interval? 1 s? 1ms? Networks vary a lot! In a local wired network, might see e2e latencies < 1ms. In a cellular network, or intercontinenal network, might see latencies > 1s.

Too short --> always resending
Too long --> underutilization of network (show)

So what should we do? Adapt timeout!

Measure round trip time (RTT) and adjust.

RTT = time between packet sent and ack received
Just use one sample? No.
Just use last time + error? No. -- too much variability, even on one network (show slides)

Use some average of recent packets.

To avoid having to keep window of averages around, can use exponentially weighted moving average:

(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:

- 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.*

(Skip?) How big to make sequence numbers?

16 bits? In gigabit net, with 1 kbit packets, will send 1 million packets / sec. So will exhaust sequence number space in about 60 ms. Packets could easily be delayed this long.

32 bits? 4000 seconds. Better, but some connections will last this long. Need to recycle sequence numbers.

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:

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.

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),

Max packets outstanding is then

window = RTT x rate

"bandwidth delay product"

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.) This is the topic for next time.

Acks and windows -- two possibilities:

- acks are per-message, as above.
    + precise (only retransmit one packet)
    - can resend data that doesn't need to be resent if ack is lost

- acks are cumulative
    e.g., indicate the sequence number up to which all packets have been seen
    + one ack can cover many packets (fewer acks)
    - sender doesn't know exactly what was lost, so has to retransmit a lot more

In practice, use both:
    - cumulative in common case (e..g, every n packets)
    - selective when there are single losses

Out or order delivery

Even though all packets arrive, packets may not arrive in order, because of windowing

Solution: re-ordering buffer.

Fixed size buffer at receiver recording out of order packets. If a packet arrives in order, deliver to app. Otherwise, add to out of order buffer while we wait for more recent packet to arrive. Diagram:

When a missing packet arrives, go ahead and send prefix of buffer to app.