Last Time

- Modularity, Layering, and Decomposition
- Example: UDP layered on top of IP to provide application demux (“ports”)
- Resource sharing and isolation
- Statistical multiplexing - packet switching
- Dealing with heterogeneity
  - IP “narrow waist” -- allows many apps, many network technologies
  - IP standard -- allows many impls, same proto

Today: TCP and Apps

- Remember from last time...
- IP service model: “best-effort”
  - Can drop, mangle, re-order, delay packets
- Easy model to provide (imposes few requirements on underlying layers--widely applicable)
- Less fun model to program to if you happen to need reliability, in-order, correct data

- Models to reason about network and system behavior
- Fail-stop, fail-stutter, byzantine failures
- Sync/Async communication model
- etc.
Design Question

- If you want reliability, etc.
- Where should you implement it?

Option 1: Hop-by-hop

Option 2: end-to-end

Options

- Hop-by-hop: Have each switch/router along the path ensure that the packet gets to the next hop
- End-to-end: Have just the end-hosts ensure that the packet made it through
- What do we have to think about to make this decision??

A question

- Is hop-by-hop enough?
- [hint: What happens if a switch crashes? What if it's buggy and goofs up a packet?]

The End-to-End Argument

If you have to implement a function end-to-end anyway (e.g., because it requires the knowledge and help of the end-point host or application),

- don’t implement it inside the communication system
- unless there’s a compelling performance enhancement

Keep in mind

- This is an engineering rule of thumb to be weighed against other design guidelines.
- Not a law.
- But in practice, it’s proved to be a nice way to think about things.
- You may encounter situations where you can’t (for whatever reason - technical, financial, political) implement things in the way the argument suggests. The real world can be an ugly place. :)

Let’s apply that to our question...

- Of where to do retransmissions.
- What does e2e argument argue for here?
- TCP uses end-to-end retransmissions.
- Can you think of times we might want to also implement hop-by-hop retransmission?
  - Hop-by-hop retransmission is cheaper and faster (count the “packet-miles” that are traveled).
  - So maybe a very high-loss link -- like wireless!
  - Your wireless card handles a few retransmissions on its own.

Rough view of TCP

(This is a very incomplete view - take 15-441.:)

Application View of TCP

- Remember socket API basics from 15-213.

What TCP does:
1) Figures out which packets got through/lost
2) Figures out how fast to send packets to use all of the unused capacity.
   - But not more
   - And to share the link approx. equally with other senders.
Socket API Operation Overview

Blocking sockets

- What happens if an application write()s to a socket waaaaay faster than the network can send the data?
- TCP figures out how fast to send the data...
- And it builds up in the kernel socket buffers at the sender... and builds...
- until they fill. The next write() call blocks (by default).
- What’s blocking? It suspends execution of the blocked thread until enough space frees up...

In contrast to UDP

- UDP doesn’t figure out how fast to send data, or make it reliable, etc.
- So if you write() like mad to a UDP socket...
- It often silently disappears. Maybe if you’re lucky the write() call will return an error. But no promises.

take a breath.
Rehashing all of that...

- TCP is layered on top of IP
  - IP understands only the IP header
  - The IP header has a "protocol" ID that gets set to TCP
  - The TCP at the receiver understands how to parse the TCP information
- IP provides only "best-effort" service
- TCP adds value to IP by adding retransmission, in-order delivery, data checksums, etc., so that programmers don't have to re-implement the wheel every time. It also helps figure out how fast to send data. This is why TCP sockets can "block" from the app perspective.
- The e2e argument suggests that functionality that must be implemented end-to-end anyway (like retransmission in the case of dead routers) should probably be implemented only there -- unless there's a compelling perf. optimization.

Questions to ponder

- What does the end-to-end argument say about where to implement encryption for confidentiality?
- If you have a whole file to transmit, how do you send it over the Internet?
  - You break it into packets (packet-switched medium)
  - TCP, roughly speaking, has the sender tell the receiver "got it!" every time it gets a packet. The sender uses this to make sure that the data's getting through.
  - But by e2e, if you have to acknowledge the correct receipt of the entire file... why bother acknowledging the receipt of the individual packets???

Answers

- 1) Encrypt end-to-end.
  - A notable exception: The military sometimes uses hop-by-hop so that they can run unencrypted on physically secure links ... so that they can monitor the traffic there.
- 2) This is a bit of a trick question -- it's not asking e2e vs in-network. :) The answer: Imagine the waste if you had to retransmit the entire file because one packet was lost. Ow.
Application Requirements

Q: If you’re building an application...

How do you choose what transport service to use?

A: That depends what the application’s communication requirements are...

What Transport Service Does an Application Need?

Data loss
- Some applications (e.g., audio) can tolerate some loss
- Other applications (e.g., file transfer, telnet) require 100% reliable data transfer

Timing
- Some applications (e.g., Internet telephony, interactive games) require low delay to be “effective”

Bandwidth
- Some applications (e.g., multimedia) require a minimum amount of bandwidth to be “effective”
- Other applications (“elastic apps”) will make use of whatever bandwidth they get

User Datagram Protocol (UDP): An Analogy

<table>
<thead>
<tr>
<th>UDP</th>
<th>Postal Mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single socket to receive messages</td>
<td>Single mailbox to receive letters</td>
</tr>
<tr>
<td>No guarantee of delivery</td>
<td>Unreliable 😎</td>
</tr>
<tr>
<td>Not necessarily in-order delivery</td>
<td>Not necessarily in-order delivery</td>
</tr>
<tr>
<td>Datagram – independent packets</td>
<td>Letters sent independently</td>
</tr>
<tr>
<td>Must address each packet</td>
<td>Must address each reply</td>
</tr>
</tbody>
</table>

Example UDP applications
Multimedia, voice over IP

Transmission Control Protocol (TCP): An Analogy

<table>
<thead>
<tr>
<th>TCP</th>
<th>Telephone Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable – guarantee delivery</td>
<td>Guaranteed delivery</td>
</tr>
<tr>
<td>Byte stream – in-order delivery</td>
<td>In-order delivery</td>
</tr>
<tr>
<td>Connection-oriented – single socket per connection</td>
<td>Connection-oriented</td>
</tr>
<tr>
<td>Setup connection followed by data transfer</td>
<td>Setup connection followed by conversation</td>
</tr>
</tbody>
</table>

Example TCP applications
Web, Email, Telnet
Why not always use TCP?

- TCP provides “more” than UDP
- Why not use it for everything?!
- A: Nothing comes for free...
  - 1) Connection setup (take on faith) -- TCP requires one
    round-trip time to setup the connection state before it
    can chat...
- How long does it take, using TCP, to fix a lost packet?
  - At minimum, one “round-trip time” (2x the latency of the network)
  - That could be 100+ milliseconds!
- If I guarantee in-order delivery, what happens if I lose one packet in a stream of
  packets?

One lost packet

Design trade-off

- If you're building an app...
- Do you need everything TCP provides?
- If not: Can you deal with its drawbacks to take advantage of the subset of its
  features you need?
- If not: You're going to have to implement the ones you need on top of UDP
  - Caveat: There are some libraries, protocols, etc., that can help provide a middle
    ground.
  - Takes some looking around - they're not as standard as UDP and TCP.

Transport Service Requirements
of Common Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Data loss</th>
<th>Bandwidth</th>
<th>Time Sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>file transfer</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>e-mail</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>web documents</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>real-time audio/video</td>
<td>audio: 5Kb-1Mb</td>
<td>yes, 100’s mseconds</td>
<td></td>
</tr>
<tr>
<td>video</td>
<td>audio:10Kb-5Mb</td>
<td>yes, few secs</td>
<td></td>
</tr>
<tr>
<td>stored audio/video</td>
<td>same as above</td>
<td>yes, few secs</td>
<td></td>
</tr>
<tr>
<td>loss-tolerant</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>interactive games</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>loss-tolerant</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>financial apps</td>
<td>no loss</td>
<td>elastic</td>
<td>yes and no</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Interactions between layers are important.
  - persistent HTTP
  - encryption and compression
  - MPEG frame types. Loss & real-time video.
Proj 1 and today’s material

- You’ll use UDP. Why?
  - A1: The course staff is full of sadists who want you to do a lot of work. This is true in part: timeouts and retransmission are a core aspect of using the network.
  - A2: The communication needed is very small, and you have to implement a lot of reliability stuff anyway to ensure that the work gets done...
  - Honestly? This one seems to me like a middle ground. You might use TCP for “other” reasons (firewalls that block everything but TCP), or to avoid the need for the “job ack” part of the protocol. Or you might stick with UDP to reduce the overhead at the server.