Week 8 - Monday

COMP 3400

Last time

- What did we talk about last time?
- UDP: DNS

Questions?

Project 2

Broadcasting

Static and dynamic IP addresses

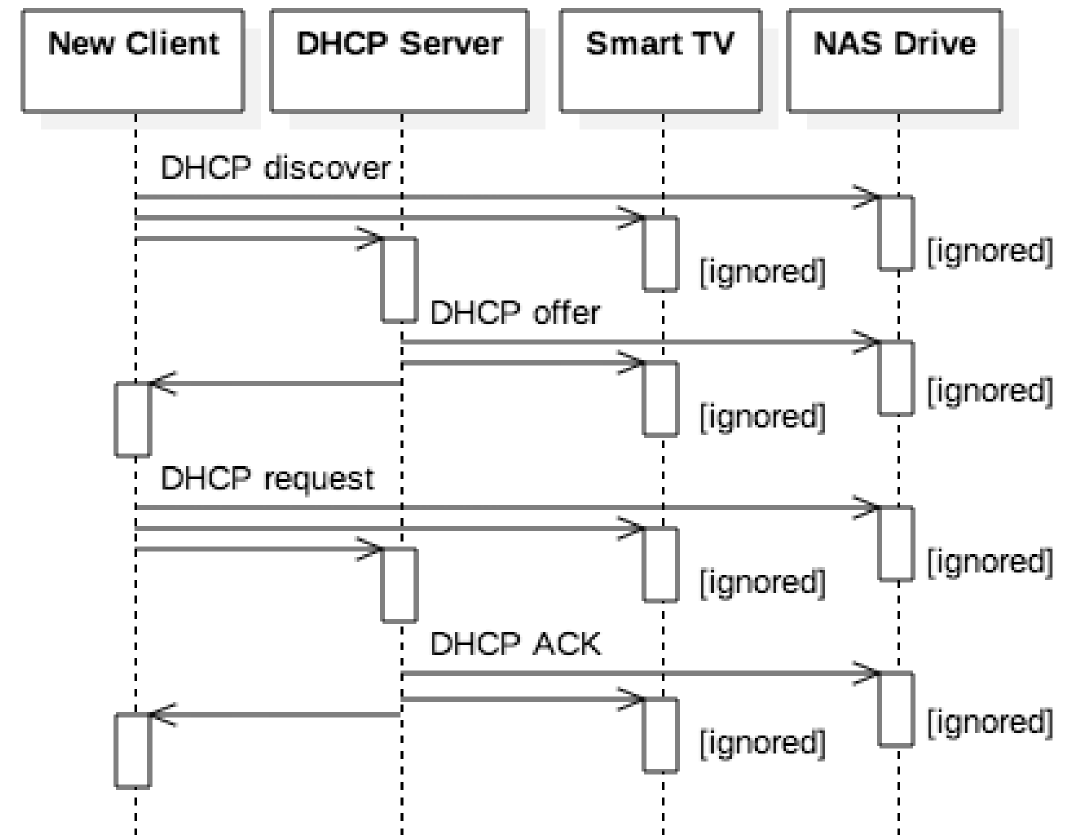
- Networked devices are configured to have either a static or dynamic IP address
  - **Static IP addresses** are set in a configuration file and rarely change
  - **Dynamic IP addresses** are assigned when a device connects to a network
- Most of the devices you own have dynamic IP addresses
  - Your laptop or phone is dynamically assigned an IP address by your router when you connect, either on WiFi or Ethernet
  - Even your router is dynamically assigned an IP address by your ISP
- Servers usually have static IP addresses so that DNS records don't need constant updates

DHCP

- But if you've just connected your device to the network, how does it get its IP address?
- **Dynamic Host Configuration Protocol** (**DHCP**) is a protocol for getting a dynamic IP address
- The socket programming we've been talking about requires an IP address for a client
  - That's where messages are sent back to
- So how do you receive a message if you don't have an IP address...because you're trying to get an IP address?
- **Broadcasting!**

Broadcasting

- Instead of sending a message point to point, DHCP messages are broadcast via UDP on port 67
  - The destination is the special IP address 255.255.255.255 reserved for broadcasting
  - The source is the special IP address 0.0.0.0 indicating no valid IP
- A DHCP server receives messages and responds with an IP address
- Other devices ignore the messages



DHCP steps

- DHCP is more complex than HTTP or DNS since it's got multiple steps
  1. The new device broadcasts a DHCP discover message asking for an IP
  2. The DHCP server broadcasts a DHCP offer message offering an IP
  3. The new device broadcasts a DHCP request message asking to take the IP it was offered
  4. The DHCP server broadcasts a DHCP ACK message acknowledging that the device has been assigned the address in question
- Like DNS, the device uses a random `xid` to keep different requests straight
- When the device requests the IP it's been offered, it increments the `xid` by `1`

DHCP example

- The table shows an example of the addresses and messages broadcast to request and assign an IP address
  - **yiaddr** is the new IP address
  - **siaddr** is the server IP address
  - The lease time is how long the IP address is valid for, in seconds: 86,400 = 24 hours
- When the lease expires, the device can ask for the IP again

Message type	UDP addressing	DHCP contents
DHCP discover	SRC: 0.0.0.0:68 DEST: 255.255.255.255:67	op: BOOTREQUEST xid: 42 yiaddr: 0.0.0.0
DHCP offer	SRC: 192.168.1.1:67 DEST: 255.255.255.255:68	op: BOOTREPLY xid: 42 yiaddr: 192.168.1.7 siaddr: 192.168.1.1 lease time: 86400
DHCP request	SRC: 0.0.0.0:68 DEST: 255.255.255.255:67	op: BOOTREQUEST xid: 43 yiaddr: 192.168.1.7 siaddr: 192.168.1.1
DHCP ACK	SRC: 192.168.1.1:67 DEST: 255.255.255.255:68	op: BOOTREPLY xid: 43 yiaddr: 192.168.1.7 siaddr: 192.168.1.1 lease time: 86400

Internet

Internet

- ARPANET was originally designed for government and the military
- As more people worked on it (and similar networks), they standardized the protocols and gave access to academics and businesses
  - Many people were sending and receiving e-mails as early as the 70s
  - It was opened up to the public in the mid 90s
- Vint Cerf first coined the term "Internet"

Image from Business Insider

Reliability

- As we have emphasized several times, one of the big differences between single-machine IPC and networked IPC is reliability
- In single-machine IPC, processes could check the OS for error codes if communication goes wrong
- Over a network, there's no OS
  - We have no way of knowing why a message wasn't received
- Writing networked applications means making a choice:
  - Rely on a lower layer to handle the problem (usually by automatically error checking and re-sending packets)
  - Take responsibility for errors at the application layer (by re-sending, showing an error message, or allowing quality to degrade)

A deeper dive into the layers

- To better understand the choices we have as developers, we're going to look more deeply at the layers:
  - Application
  - Transport
  - Internet
  - Link
  - Physical

Application Layer

Peer-to-peer applications

- We have already given examples of client-server applications
  - HTTP
  - DNS
- Although some client-server interactions are much more complicated, many of the same principles will apply
- **Peer-to-peer applications** (**P2P**) are the other major, application-layer approach
  - Every host is potentially a client and a server
  - Communicating with peers is more complex because there isn't a single server to keep track of
- In many situations, P2P applications can provide better performance than client-server when the number of hosts is large

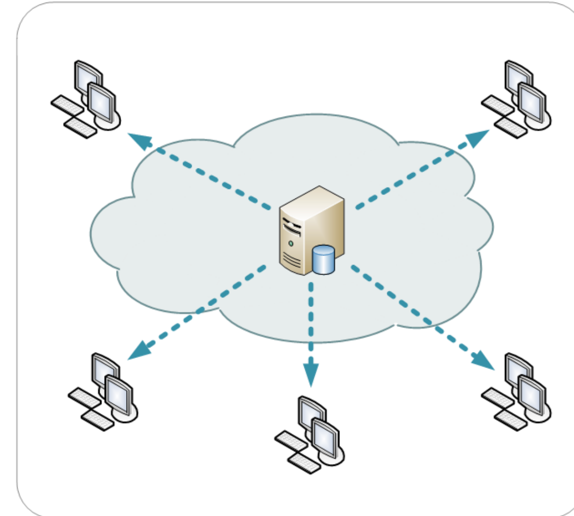
P2P examples

- P2P has a historic association with illegal file sharing, but P2P architectures are used for many different kinds of applications

Application	Service Description	Examples
Content Distribution	Scalable approaches to sharing data across the Internet	▪ **File storage and sharing:** Gnutella, BitTorrent, InterPlanetary File System (IPFS) ▪ **Content delivery networks (CDNs):** Akamai, Limelight ▪ **Streaming media:** Spotify (originally), Sonos ▪ **Software update distribution:** Linux, World of Warcraft
Distributed Computing	Delegating work for an application across many computers	▪ **Privacy and censorship resistance:** Tor, Freenet ▪ **Cryptocurrency:** Bitcoin ▪ **Botnets and malware:** Storm
Collaboration	Providing real-time human communication	▪ **Voice Over IP (VOIP):** Skype (originally) ▪ **Instant Messaging:** Tox
Platforms	Building applications	▪ **Java:** JXTA (obsolete)

Content delivery networks

- Normally, there's a single server for a webpage
- What if that webpage has content that millions of people from all over the country want to view?
  - The load on the server will be huge
  - Getting the webpage will be slow, or the server could crash
- Content delivery networks (CDNs) provide caches of webpages
- People trying to view a webpage will be redirected to a physically close mirror
- Big companies like Google, Amazon, and Netflix have their own CDN services
- Less well-known companies like Akamai provide CDN services to others



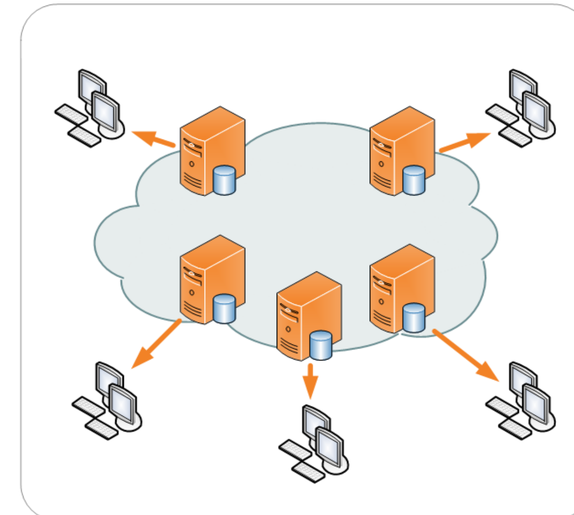
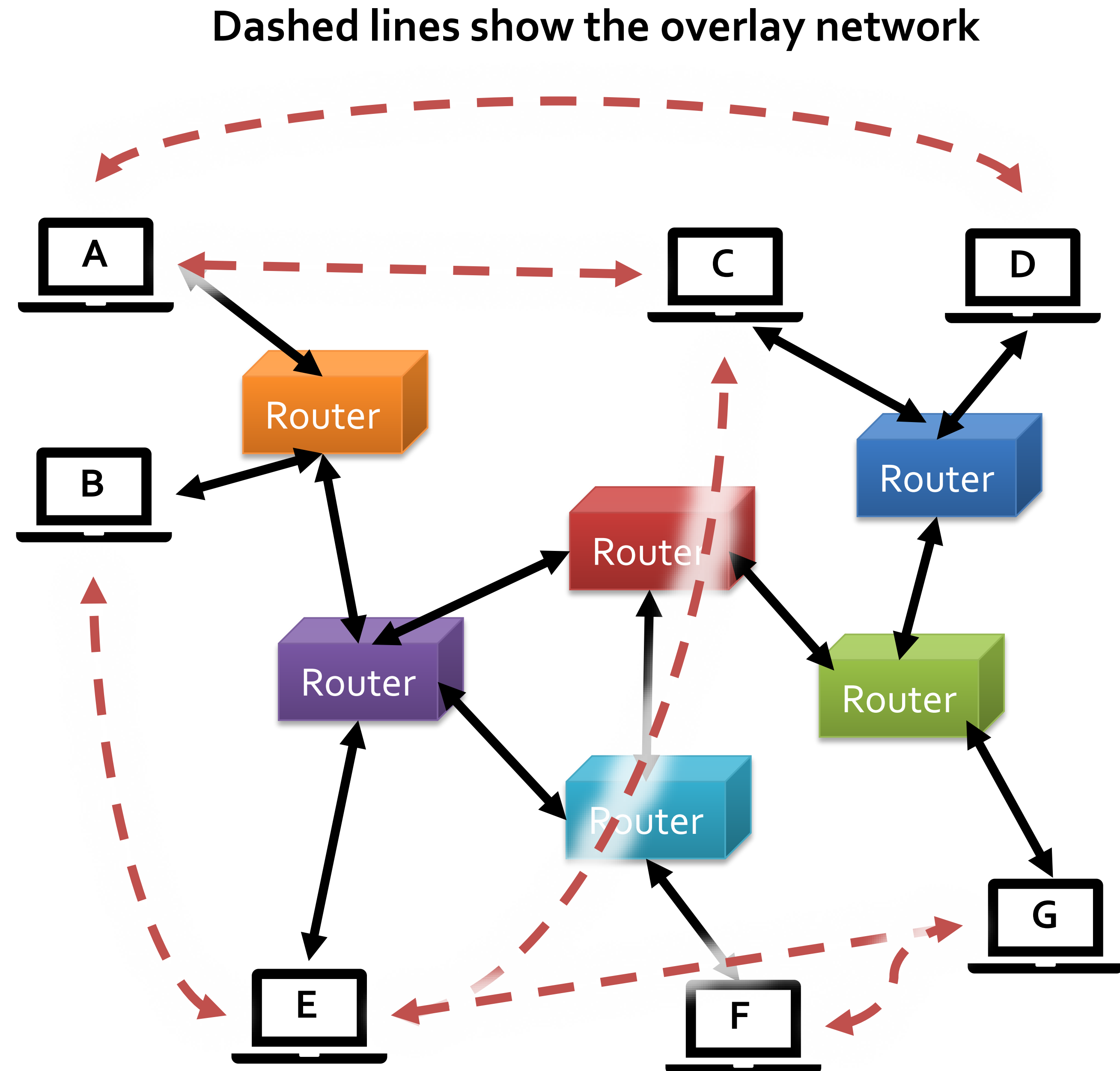


Image from [Wikipedia]

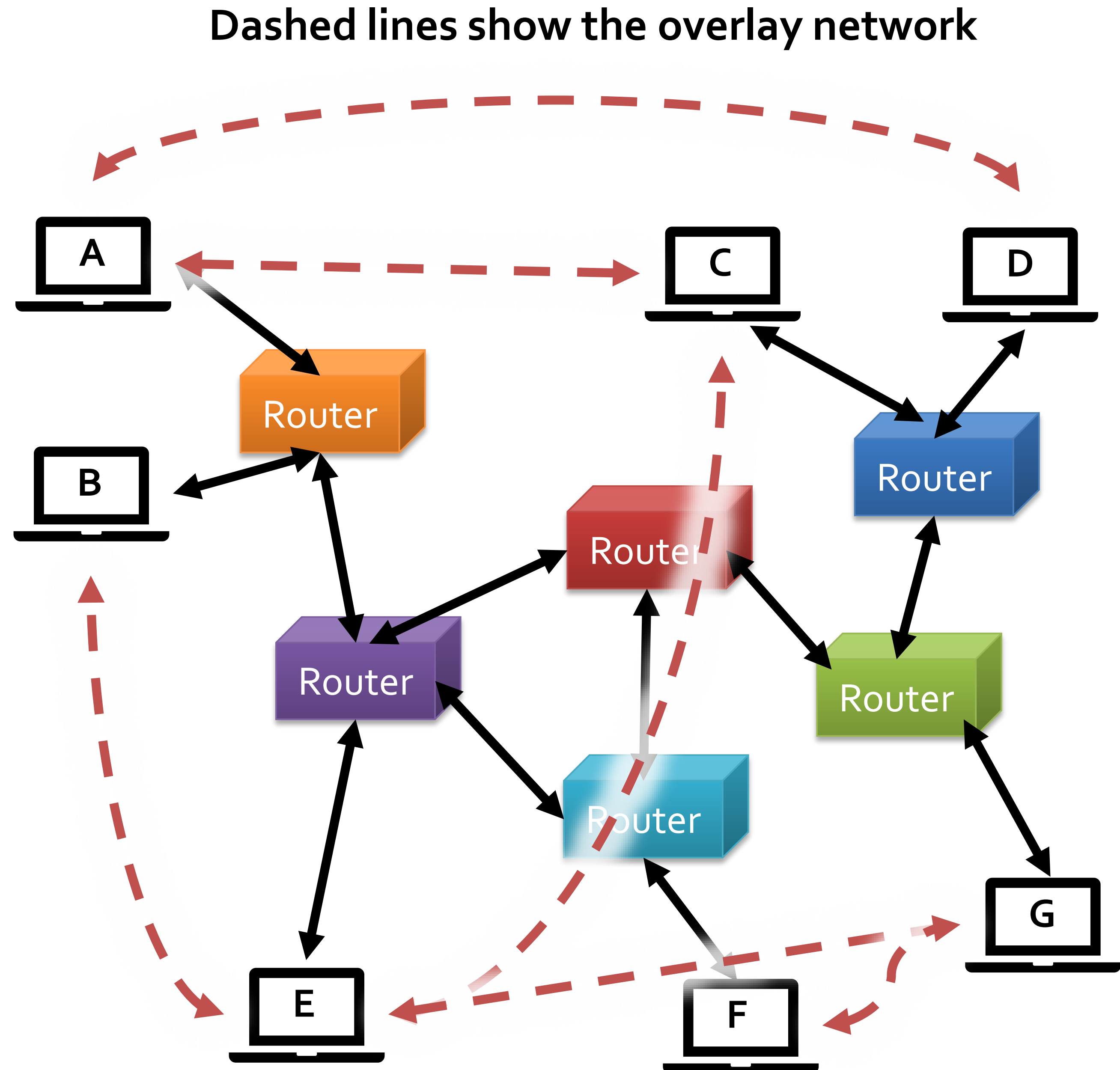
Overlay networks

- P2P hosts are fundamentally connected by the Internet
- However, they view their connections inside the P2P network as an **overlay network**, connections to other P2P hosts
- Thus, socket connections are made to P2P neighbors who forward messages on to other neighbors



Overlay networks

- For A to send a message to B, it has to send it to C, which sends it to E which sends it to B
  - Even though A and B are on the same network!
- While this arrangement seems inefficient, it can be used in applications like Tor, where the goal is to hide the true addresses of the hosts



Characteristics of P2P networks

- **Design decision:** Should a P2P network be structured or unstructured?
- Early P2P networks were often unstructured
  - This approach made sense for illegal file sharing
  - Unstructured networks have a lot of **churn**, hosts arriving and leaving frequently
- Many P2P networks are now structured with a logical framework
  - Maybe the overlay network arranges nodes in a circle, with each node knowing about the node before it and after it
  - A CDN might have an organization based on physical proximity

More characteristics of P2P networks

- **Design decision:** How are objects like files identified in the network?
- Unstructured networks often use **query flooding**
  - Ask all your neighbors if they have a file
  - If they don't have it, they'll ask their neighbors, and so on
- Structured networks have more options
  - Indexing objects based on location
  - Local indexes only know about neighbors
  - Depending on the structure, algorithms can be used to search the network
  - Centralized indexes are simple but put strain on a central server
  - Other approaches distribute the index across several servers

Transport Layer

Transport layer

- The transport layer provides a logical structure for end-to-end communication between two different (networked) processes
- Although there are other protocols for the transport layer, the most common ones are flavors of UDP and TCP
- As we have pointed out in the past:
  - TCP provides reliable transport that tries to fix failures
  - UDP is faster but unreliable

UDP

- The **User Datagram Protocol** (**UDP**) provides a bare-bones approach to sending messages
- Information included in a UDP segment is:
  - Source port
  - Destination port
  - Length of the segment
  - Checksum
  - Payload (actual data)
- Each header field is 16 bits, making a header of 8 bytes for each UDP segment in addition to the data

Checksum

- UDP uses a checksum to make sure that the segment isn't corrupted during transmission
- It is possible (but unlikely) that a message with some bits flipped will have the same checksum as the original
- Pseudo-code:
  - Add up all the 16-bit quantities in the message into a 32-bit sum
  - While adding, if the most significant bit of the sum is 1, change the sum to be the sum of its lower and upper halves
  - If there was an odd number of bytes, add the last byte padded with zeroes
  - After the sum is made, add the lower and upper halves of it to get a 16-bit value
  - Return the bitwise negation of the result

Actual code for checksum

- The following code computes the checksum for data
- Note that a 16-bit pointer is given to the data, dealing with two bytes at a time
- Even so, the length is the number of bytes (8-bit quantities)

```
uint16_t cksum (uint16_t *bytes, size_t length)
{
  uint32_t sum = 0;
  for (size_t i = 0; i < length / 2; i++) // Loop through each chunk of 16 bits
    {
      sum += bytes[i];
      if (sum & 0x80000000) // If there's a leading 1, add the two halves
        sum = (sum & 0xffff) + (sum >> 16);
    }
  if (length % 2 == 1) // For an odd number of bytes, pad the last with zeroes
    sum += ((uint8_t *)bytes)[length - 1];

  // Combine the two halves and flip the bits
  sum = (uint16_t) sum + (uint16_t) (sum >> 16);
  return ~sum;
}
```

Example UDP segments

- The following are UDP segments for a DNS request and response

Header	1388 0035 0025 f693	source port = 5000 (0x1388) destination port = 53 (0x0035) length = 37 (0x0025) checksum
Payload	1234 0100 0001 0000 0000 0000 0765 7861 6d70 6d64 0363 6f6d 0000 0100 01	DNS request for **example.com**

Header	0035 1388 0035 af04	source port = 53 (0x0035) destination port = 5000 (0x1388) length = 53 (0x0035) checksum
Payload	1234 8180 0001 0001 0000 0000 0765 7861 6d70 6d64 0363 6f6d 0000 0100 01c0 0c00 0100 0100 00e9 4900 045d b8d8 22	DNS response for **example.com**

Unreliability

- Once the UDP segment arrives, the receiver can compute a checksum for the segment to see if it matches the one provided
- UDP itself doesn't do anything with this checksum value, but the applications built on UDP can decide to request the data again or ignore the bad data
- This lack of reliability seems like a problem, but it can be useful for streaming movies or audio
- It's also useful for DNS and DHCP, which are not usually visible to the user

Upcoming

Next time…

- Finish TCP
- Network security

Reminders

- Work on Project 2
  - **Due Friday by midnight!**
- Read section 5.4