Week 9 - Monday

COMP 3400

Last time

- What did we talk about last time?
- Network security
  - Symmetric cryptography
  - Public key cryptography
  - Cryptographic hash functions

Questions?

Assignment 5

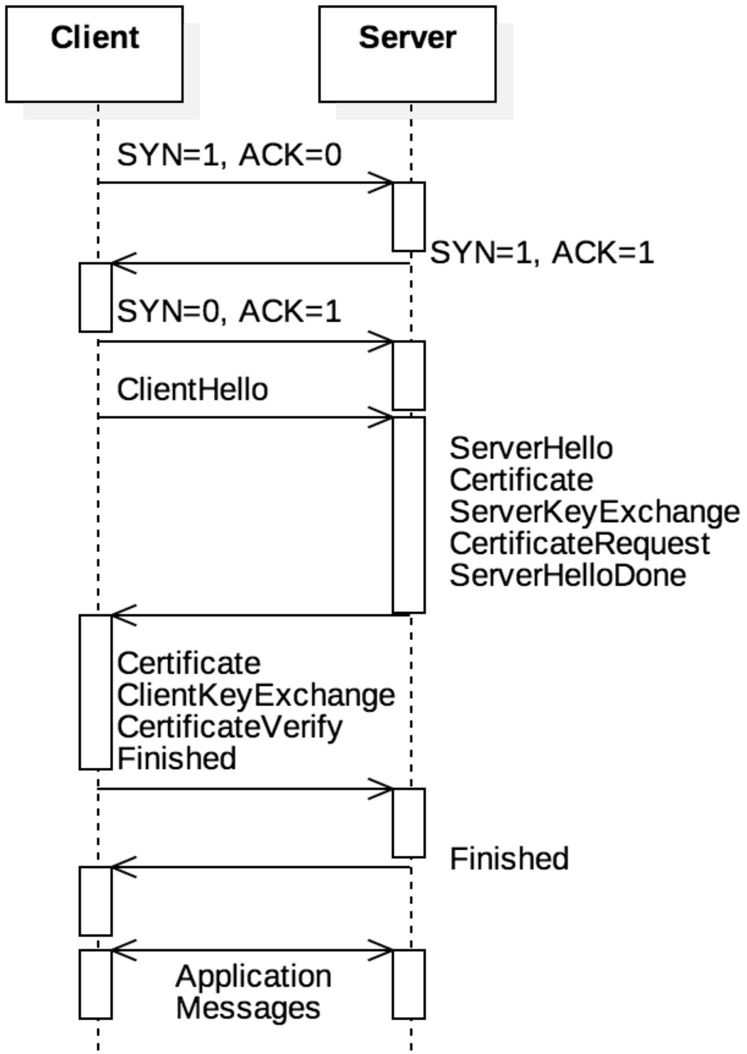
TLS

Transport-Layer Security

- **Transport-Layer Security** (**TLS**) adds end-to-end security to TCP
  - Secure versions of protocols often add an "s" to their names: HTTPS, SFTP, and IMAPS
  - These protocols use TLS
- With TLS, the TCP data is encrypted
- However, the TCP headers are *not* encrypted
  - If they were, the OS wouldn't know which port to deliver them to
  - Because network traffic needs to know where to go, it's usually possible to do traffic analysis, even when the data is encrypted

TLS handshake

- With TLS, the connection first performs a TCP three-way handshake
- Then, the client and the server perform a TLS handshake that uses public key cryptography to agree on a session key
- The session key is used to communicate securely using symmetric key encryption (probably AES) during the TCP session



Confidentiality and integrity

- Because the data in the TCP segments is encrypted with AES, the information's confidentiality is maintained
- To protect integrity, a message authentication code (MAC) of the TCP headers is attached as an optional TCP field
  - The MAC is a cryptographic hash digest, probably using SHA-2
- These are the broad strokes, but there are many details
- Details change with each version of TLS
  - We're up to TLS 1.3 now

Security is hard

- TLS is the successor to SSL
- SSL had three versions but was eventually replaced by TLS because of security flaws
- Security flaws exist in TLS 1.0, 1.1, and 1.2, leading to the adoption of TLS 1.3
- In some cases, the flaws are because encryption algorithms that were discovered to be insecure are allowed
- In other cases, the protocols themselves had vulnerabilities that could be exploited
- **Key takeaway:**
  - Security is not one-and-done
  - Application security should be designed so that it's easy to change over to newer standards

Internet Layer

Internet layer

- TCP and UDP provide a framework for end-to-end communication between *processes*
- But they ignore the fact that different *hosts* are communicating
- The Internet layer provides a system for getting messages from host to host
  - The **data plane** gives the structure of the network, using Internet Protocol (IP) addresses
  - The **control plane** controls how messages are routed through the network

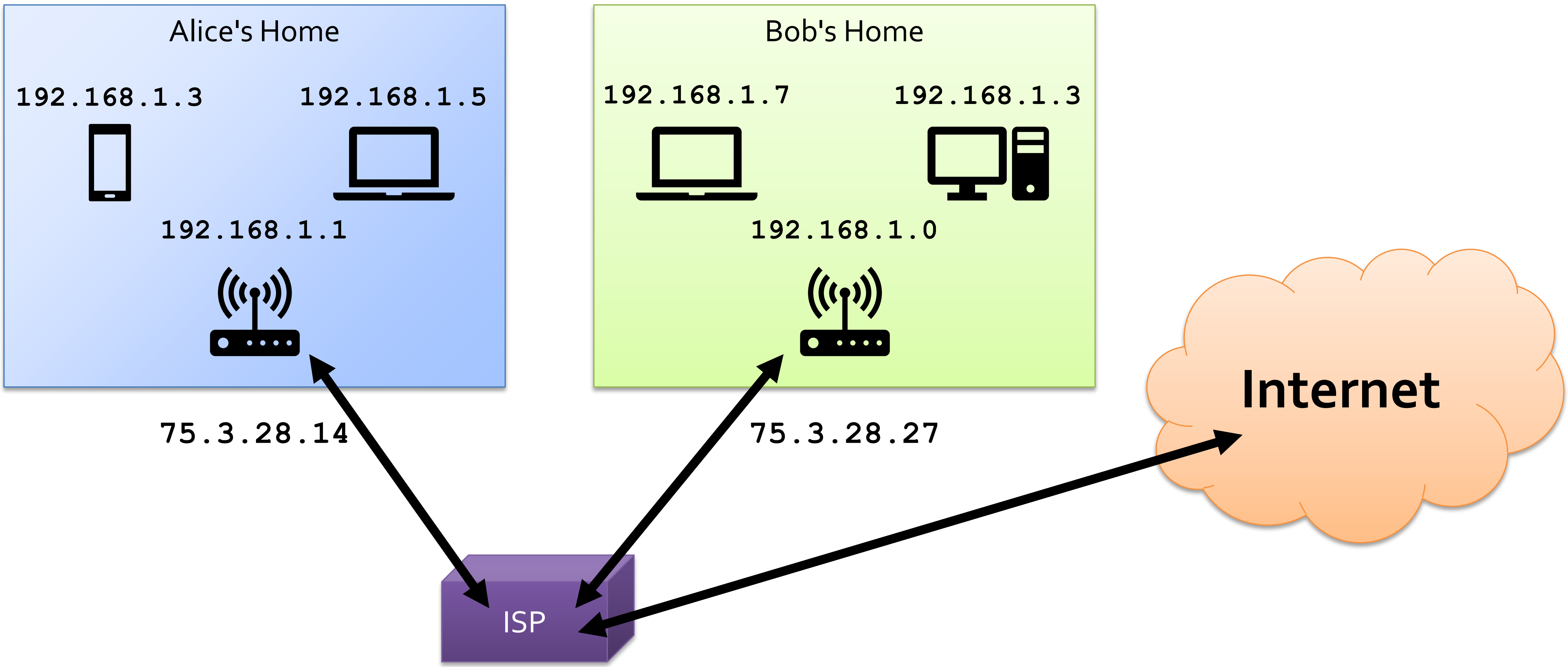
Special subnets

- Three ranges of IP addresses are reserved in IPv4 for private subnets:
  - **`192.168.0.0/16`** (2^{16} = 65,536 possible devices)
  - **`172.16.0.0/12`** (2^{20} = 1,048,576 possible devices)
  - **`10.0.0.0/8`** (2^{24} = 16,777,216 possible devices)
- The first range is probably familiar to you because it's used for most home networks
- IPv6 has its own range, **`fd00::/8`**, that allows for up to 2^{64} devices

NAT

- So that different subnets can communicate, a router connects the private subnet to the Internet
  - The router has a private IP address, used to communicate with the subnet, and a public IP address, used to communicate with the rest of the world
- Routers do **network address translation** (**NAT**), a kind of **IP masquerading**
  - The outside world only sees the router's IP
  - When the router gets a message, it sends it to the appropriate device in the private subnet
  - The router observes traffic and changes port numbers on incoming and outgoing packets so that multiple devices behind the router can communicate with a single server

Visualization of subnets



IPv4 packet format

- **version** distinguishes between IPv4 and IPv6
- **protocol** is TCP or UDP
- **checksum** is just for the header and does no checking for the payload
- **TTL** gives the number of times the packet can be forwarded (keeps packets from hopping around forever)
- Like UDP, IP makes no guarantees about reliability
- The purple **options** fields are variable length

0-3	4-7	8-11	12-15	16-19	20-23	24-27	28-31
version	length	type of service		total length			
identification				flags	fragment offset		
TTL		protocol		checksum			
source address							
destination address							
options							
payload (transport-layer segment)							

IP packet example

- Here's an example of the values (in hex) that might be stored in an IPv4 packet
- Note that IPv6 packets are similar but simpler, because they don't have optional fields

Header	`4500`	`IPv4, length = 20 bytes (5 words)`
	`0060`	`total length = 96 bytes`
	`0000 0000`	`ID, flags, offset (not used here)`
	`08`	`TTL`
	`06`	`protocol (TCP)`
	`6862`	`checksum`
	`867e 8ddd`	`source address 134.126.141.221`
	`5dd8 d822`	`destination address 93.184.216.34`
Payload	…	

Network routing protocols

- The Internet is a network of networks
  - Each independent network controlled by a single entity is called an **autonomous system** (**AS**)
- Each AS connects to other ASes at gateway routers
  - BGP is a protocol that describes how these routers communicate to each other the paths through them to other networks
- Within an AS, OSPF, RIP, and other protocols determine the fastest route through the network
  - OSPF uses Dijkstra's shortest path algorithm based on time delays, broadcasting information to other routers
  - Alternatively, when a router using RIP discovers a new shortest path, it forwards the information only to its neighbors

Link Layer

Link layer

- The Internet layer focuses on routing packets through networks
- The **link layer** focuses on forwarding packets from point to point
- This forwarding all happens within a single kind of technology
- Things can go wrong at this fundamental level of networking:
  - **Processing delay** because checksums and other information have to be computed
  - **Queuing delay** because other packets are waiting to be sent
  - **Transmission delay** because converting to the physical layer takes work
  - **Propagation delay** because the physical layer can't send data instantly
- All these delays can add up

Ethernet

- Ethernet is one of the best known examples of link level protocols
- Ethernet is a collection of standards for communicating over copper or fiber optics
- Like higher level protocols, Ethernet also wraps its data with a header (and a footer too)
  - Typically, link layer packets are called **frames**
- For historical reasons, Ethernet frames are described in **octets** (always 8 bits) rather than bytes (which used to be variable in size)

Ethernet frames

- An Ethernet frame uses:
  - 8 octets for a preamble that's always the same, to mark the start of a message
  - 6 octets for destination address
  - 6 octets for source address
  - 2 octets for type of Ethernet
  - A payload of variable size
  - 4 octets for a **cyclic redundancy check** (**CRC**), an error checking value computed from the whole frame that is stronger than a checksum
- Source and destination addresses are **media access control** (**MAC**) addresses that are usually the same for a device's entire life
- **Address Resolution Protocol** (**ARP**) is used to ask devices on the network for their MAC based on their IP

Size	8 octets	6 octets	6 octets	2 octets	varies	4 octets
Purpose	Preamble	Destination	Source	Type	Payload	CRC
Example	`aaaaaaaaaaaaaaab`	`f0def12cc22b`	`f45c89bd332d`	`0800`	`...`	`64713722`

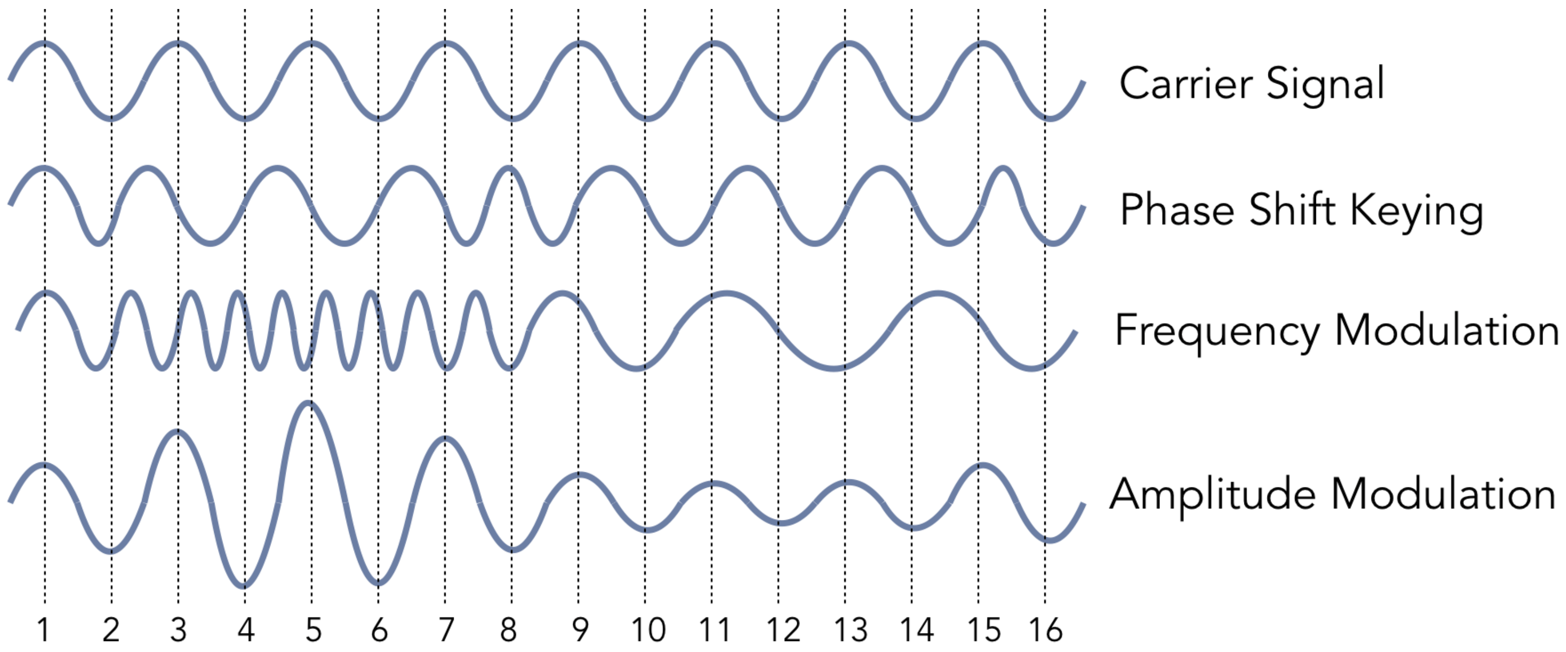
Stacks on stacks on stacks

- Here's an example of all the layers together:

<table>
<tr><td>Ethernet header</td><td colspan="3"><code>aaaa aaaa aaaa aaab f0de f12c c22b f45c 89bd 332d 0800</code></td><td></td></tr>
<tr><td>IPv4 header</td><td><code>4500 0060 0000 0000 0806 6862 867e 8ddd 5dd8 d822</code></td><td></td><td></td><td rowspan="4">Ethernet Payload</td></tr>
<tr><td>TCP header</td><td><code>1388 0050 0000 0017 0000 002a 5010 1000 cf33 0000</code></td><td></td><td rowspan="3">IPv4 Payload</td></tr>
<tr><td>HTTP header</td><td><code>4745 5420 2f20 4854 5450 2f31 2e31 0d0a 486f 7374
3a20 6578 616d 706c 652e 636f 6d0d 0a43 6f6e 6e65
6374 696f 6e3a 2063 6c6f 7365 0d0a 0d0a</code></td><td rowspan="2">TCP Payload</td></tr>
<tr><td>Ethernet FCS</td><td><code>6471 3722</code></td></tr>
</table>

A glimpse at the physical layer

- Below the link layer, the physical layer is actually communicating bits
- Bits are communicated as waves of light or radio signals, through air, fiber optics, or copper
- There are different ways of carrying a signal in a wave
- Deeper than that requires us to talk about more physics and electrical engineering than we want to right now



Wireless

- Wireless communication differs from wired at the link and physical layers and sometimes above
- There are a few important wireless network technologies:
  - **Wi-Fi** is a set of standards designed to replace normal wired networking connections
  - **Bluetooth** is designed for short-range mobile ad hoc networks (MANETS)
    - Uses a star topology where many peripherals connect to a central devices
  - **Zigbee** uses a wireless mesh network for communicating between many low powered devices
    - Popular for Internet of Things (IoT) applications

Threading

Threads and processes

- Many processes can run concurrently
  - Each one executes independently
  - Each process has its own memory layout
- Many threads can also run concurrently
  - Each one executes independently
  - Each thread has its own stack to keep track of its function calls
  - But all threads within a process **share** code, data, heap, and kernel segments
- Just as we used **`fork()`** to spawn new processes, there are libraries to spawn new threads within a process and coordinate them

Advantages of threads

- Using threads allows for more modular software since threads can call the same functions within a program
- Threads can be more efficient since there's no context switch needed for different threads to interact
- Some models of programming like GUIs depend on threads so that one unit of code needs can react to an action taken elsewhere
- Since threads share memory, there's no need for IPC libraries

Disadvantages of threads

- Threads are less isolated from each other than separate processes
- Consequences:
  - A thread crashing from a segmentation fault will kill the entire process, including the other threads
  - Bugs called **race conditions** occur, where the behavior of the program is different depending on which thread executed first

Race conditions

- Race conditions are a central problem with threads
- Thread scheduling is non-deterministic
  - It's often impossible to predict when the statements from one thread are going to be executed with respect to those in another thread
  - If the statements modify the same memory, the results can be inconsistent
- One of the most frustrating issues with race conditions is that they can occur rarely
  - This means that you can run your program 1,000 times with no problems, only to crash badly on time 1,001

Race condition scenarios

- The following are common causes of race conditions:
  - Two or more threads trying to modify a global variable at the same time
  - One thread calls `free()` on data that another thread is using
  - Thread A is using variables declared on the stack of Thread B, which become invalid when Thread B terminates
  - Two or more threads calls a non-thread-safe function at the same time

Upcoming

Next time…

- Thread safety
- POSIX threads

Reminders

- Work on Assignment 5
  - Due Friday by midnight
- Read sections 6.4 and 6.5
- Study for Exam 2
  - Next Monday!