

# Genome 540 full lecture

March 11th, 2025  
Joe Min

# Agenda

Homework 9 overview

Connecting to a website

Moving data over established connections

Network vulnerabilities

## Homework 9

## Homework 9

Part 1: Implement a 2-state HMM with empirically-derived parameters where the states correspond to evolutionarily conserved or neutral regions

Part 2: Use your HMM to find the Viterbi parse for a 3-sequence alignment of dog, mouse, and human genomic sequences

# Part 1: empirically-derived parameters

## State 1: neutrally evolving

- Using the provided list of emission counts from a large set of ancient repeat sequences, calculate emission probabilities
  - Column 1 is human
  - Column 2 is dog
  - Column 3 is mouse

AAA	10222095
AAC	481243
AAT	420185
AAG	1415675
AA-	273456
ACA	852624
ACC	179459
ACT	99493
ACG	167810
AC-	29636

# Part 1: empirically-derived parameters

## State 1: neutrally evolving

- For each emission, count the observed symbol then divide that by the total counts
  - E.g., for 'AAA', this would be:  
 $10,222,095 / \text{TOTAL}$
- These are your emission probabilities for the neutral state of your HMM

AAA	10222095
AAC	481243
AAT	420185
AAG	1415675
AA-	273456
ACA	852624
ACC	179459
ACT	99493
ACG	167810
AC-	29636

# Part 1: empirically-derived parameters

## State 2: evolutionarily conserved

- Do the same thing but using counts from a 3-sequence alignment from putative functional sites
- Emission symbols corresponding with conservation (e.g., 'AAA' or 'CCC') should have higher probabilities here compared to their probabilities in State 1

AAA	2375583
AAC	21337
AAT	10886
AAG	56328
AA-	3205
ACA	33210
ACC	12122
ACT	2270
ACG	5187
AC-	374

# Part 1: Making the initial HMM

State 1 is neutral

State 2 is conserved

Initiation probabilities

- $\pi_1$ : 0.95
- $\pi_2$ : 0.05

Emission probabilities

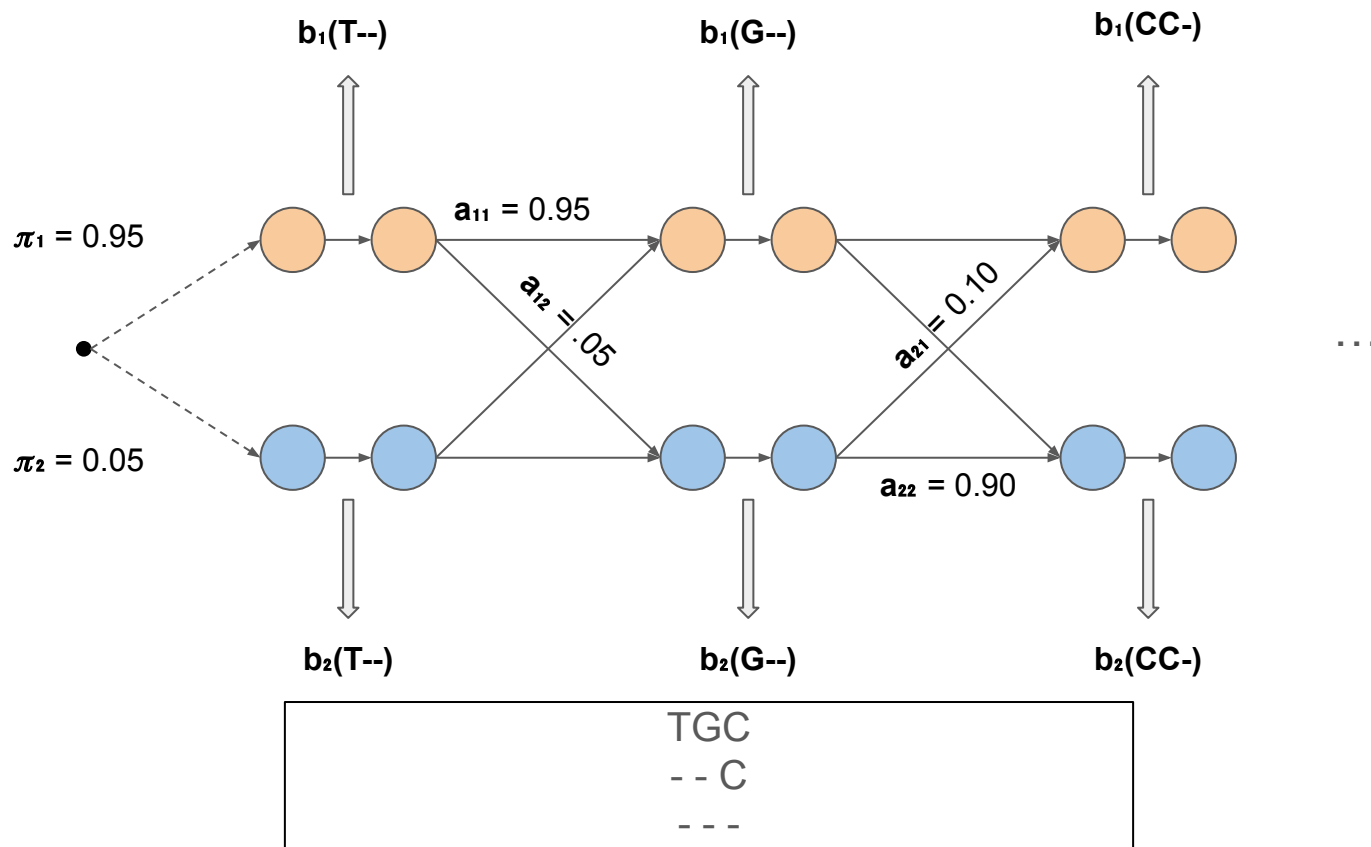
- Calculated using counts

Transition probabilities

	to State 1	to State 2
From State 1	$a_{11} = 0.95$	$a_{12} = 0.05$
From State 2	$a_{21} = 0.10$	$a_{22} = 0.90$



## Part 2: using the HMM



## Part 2: the Viterbi parse

Now, using dynamic programming, determine the highest probability state sequence for the observed alignment

```
# chr7:26924045-26924056
```

```
hg18    TGCTCACATTTT
canFam2  --CTCACAGTTT
mm9      -----CGCTT-
```

```
# chr7:26924057-26924120
```

```
hg18    CTAGAAGGATTAATGTTCTGTAGATCTATTGATCTTCTACATTCTTCTTAAAGTATCCAGGGTA
canFam2  TCAGAGGGATTAGTGTCTGTGGATCTATTGATCTTCTGCACTCTTCTAAAGTATCTGGGGGA
mm9      CCAGAGGGAGTGGTGTCTGTAGATCTATCGACCTC--CACGCAGCTAAAGTACCTCGGGTG
```

```
# chr7:26924121-26924289
```

```
hg18    ATCATTAACAATACITTTGTTTTGATTTACTTGCCTGGTGTCTGAGGCTCTCCAGCTCTCTACAATACATTTGCGCTTTATTTCATGATGCTTATTCTGTAGATAAAGACAGCACATTACTGGCATTGTAACTGGGAGGCTTAAAAATTTTAAACATAAAATTAGAGAT
canFam2  ATCATTAGCAACACATTTTGTTCTGATCTACTTGCCTGTCTATCCAAGGCTATTCAGCTCTCTAAAAATACATTTGTGCTTTATTTCATGCTTATTCTATA-ATAAAGACCTTACCTTACTGGCATTATAACTGGGAGGCATAAGACTTTTAAAAATTAGATTATATGT
mm9      -----ACTTCGCTCTGCTCCACTTGCCTGACATCCAAGGCTCTCCAGTTCTGTATAATGTCTTCGTGTTTCATTCACTATTCTTATGTTATA----AAGACTGAGTGTTCTTGGCACCTTCAATTGGAAAGCTTAA----TCAAAAAAGTAGAT-----
```

```
# chr7:26924290-26924313
```

```
hg18    AATCTAATGTTTAGATTAGGGTTA
canFam2  -----
mm9      -----TTAGA-----TA
```

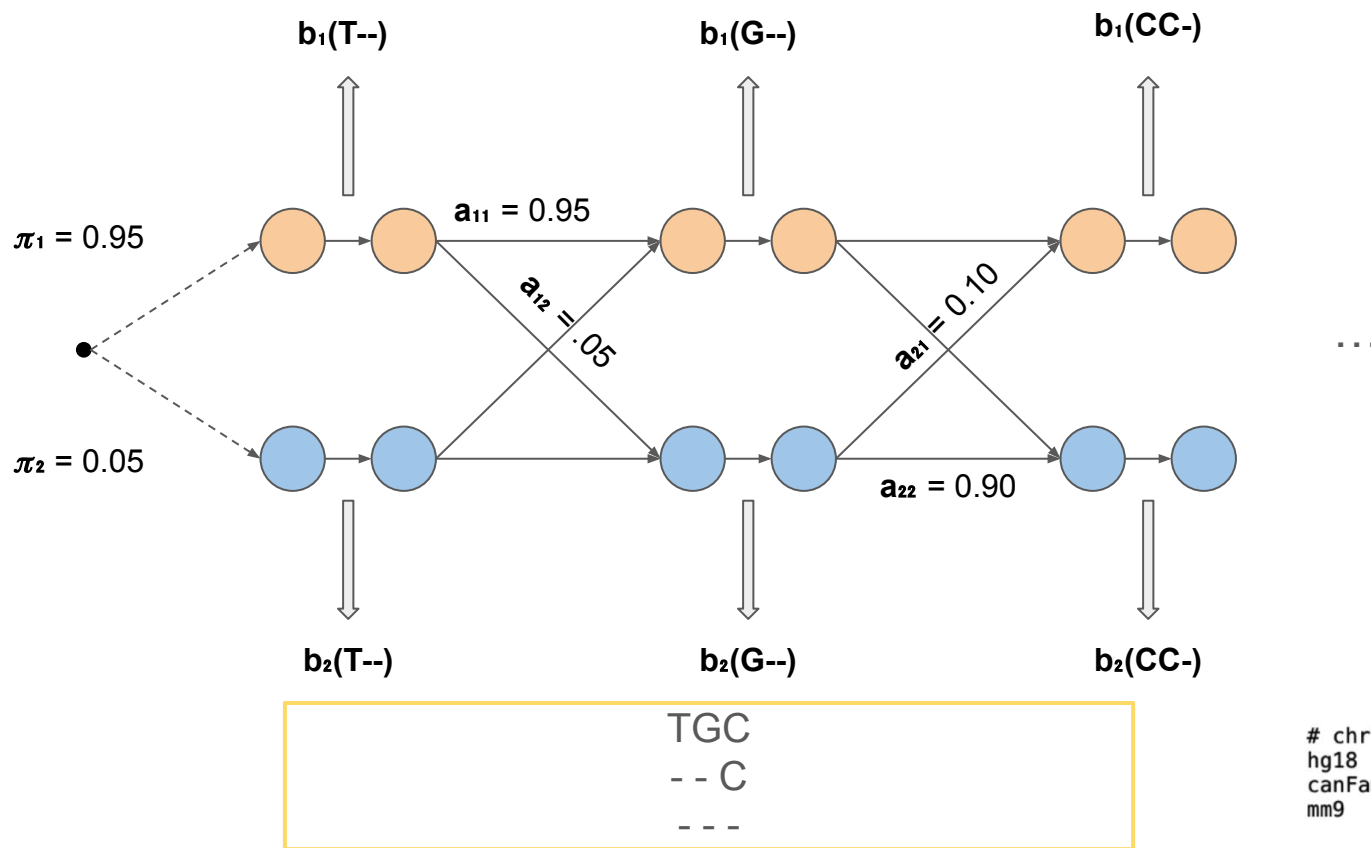
```
# chr7:26924314-26924339
```

```
hg18    GATTTTTAAATAGGGTATAGAACTTC
canFam2  GCCTTTTAAAGTAGGGGTAGATTTTC
mm9      -CCTTTTAAATGAGACACAGATCTTC
```

```
# chr7:26924340-26924382
```

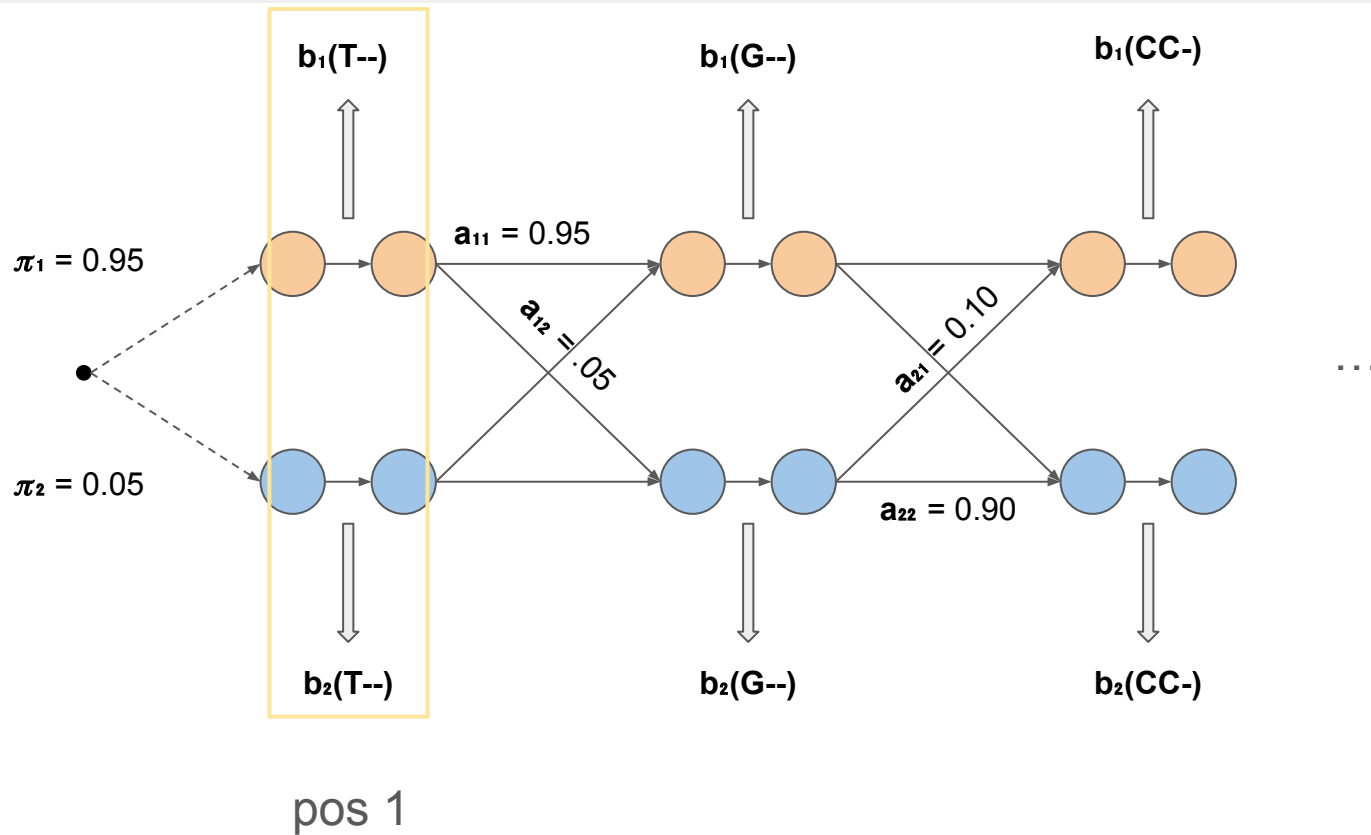
```
hg18    AAAAGAAGGTGGTTTTCTCTTTCCCTGGAAATATTCAGAAAAC
canFam2  AAGACAAAGTGGTCTTCTCTTTCCCTGGAAATACTCAGAAAGAG
mm9      ATGAGAAAGGAGACTTCTCTTTCCCTGGAAATAG-----
```

## Part 2: using the HMM

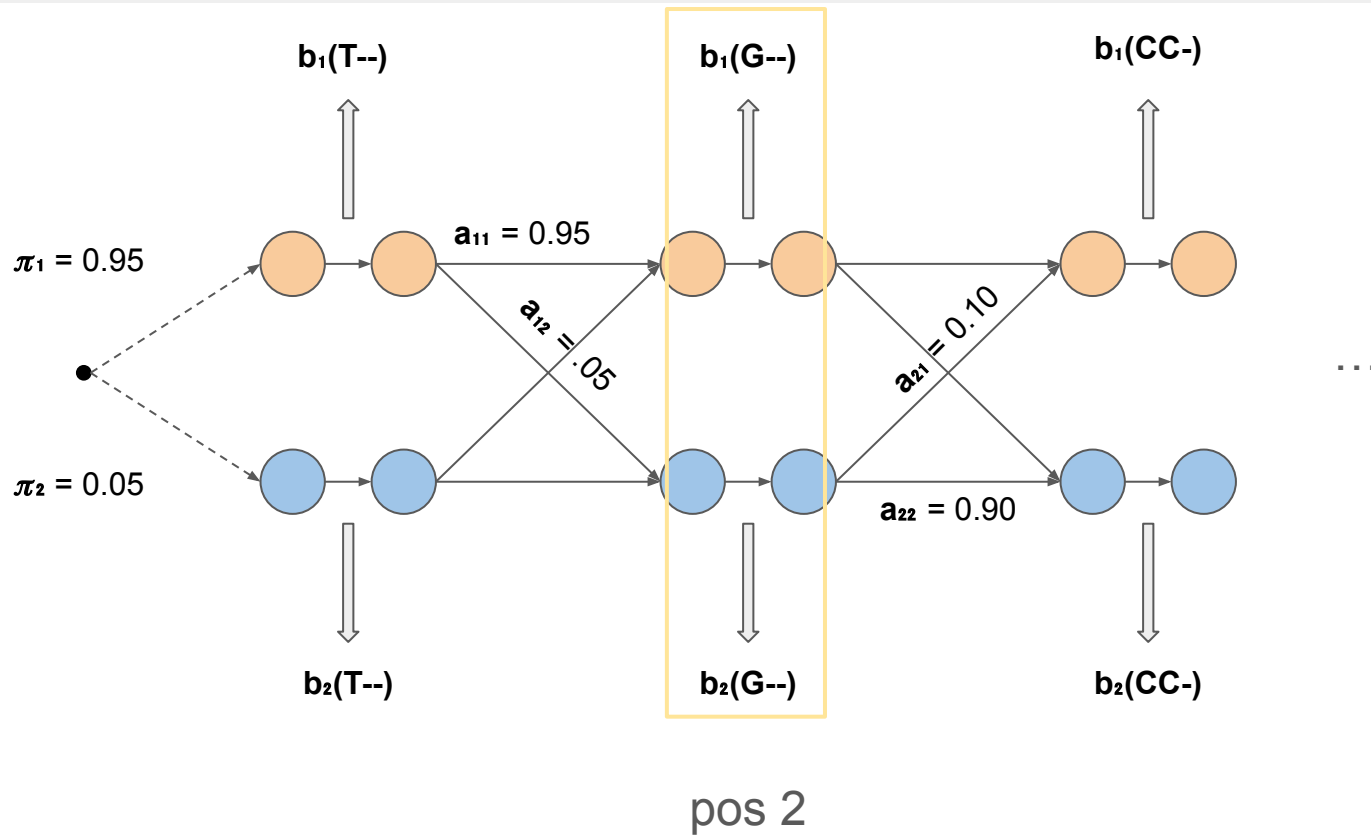


# chr7:26924045-26924056  
hg18 TGCTCACATTTT  
canFam2 --CTCACAGTTT  
mm9 ----CGCTT-

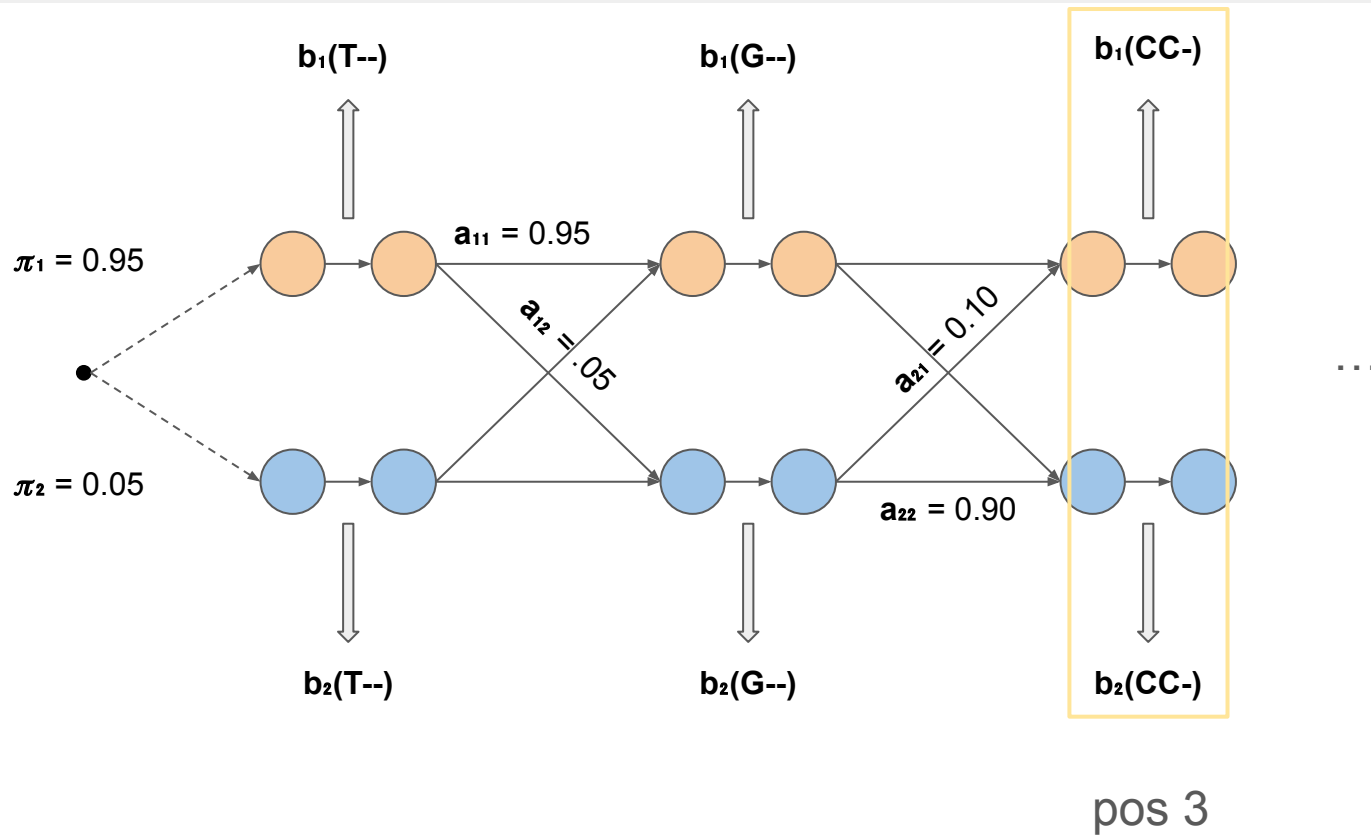
## Part 2: using the HMM



## Part 2: using the HMM



## Part 2: using the HMM



## Homework 9: output format

State histogram that counts how many times we were in each state

- E.g., if the state sequence is “1 1 2 1 1”, we should report:

**State Histogram**

**1=4**

**2=1**

## Homework 9: output format

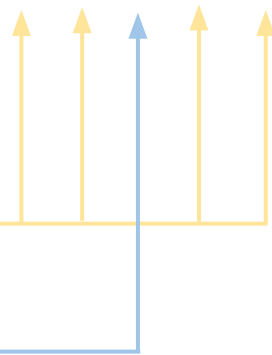
Segment histogram that counts how many contiguous segments of the same state we have

- E.g., if the state sequence is “1 1 2 1 1”, we should report:

**State Histogram**

1=4

2=1





## Homework 9: output format

Segment histogram that counts how many contiguous segments of the same state we have

- E.g., if the state sequence is “1 1 2 1 1”, we should report:

**Segment Histogram**

**1=2**

**2=1**

## Homework 9: output format

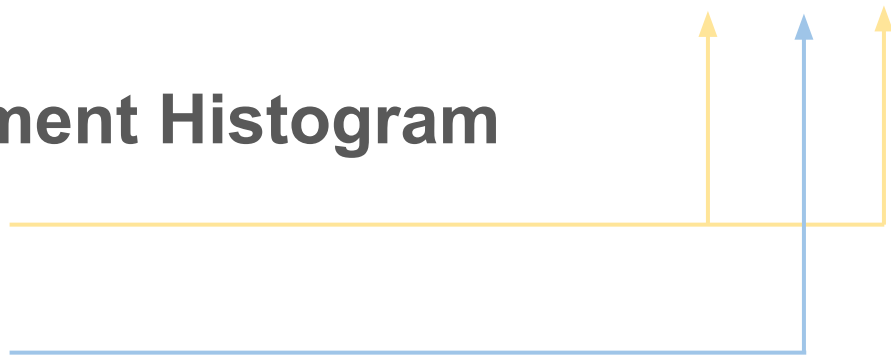
Segment histogram that counts how many contiguous segments of the same state we have

- E.g., if the state sequence is “1 1 2 1 1”, we should report:

### Segment Histogram

1=2

2=1



# Homework 9: output format

Initial state probabilities and transition probabilities (the ones given at the beginning)

Emission probabilities for all observed symbols in alphabetical order

- First for state 1, then for state 2

Initial State Probabilities:

1=0.90000

2=0.10000

Transition Probabilities:

1,1=0.99000

1,2=0.01000

2,1=0.20000

2,2=0.80000

Emission Probabilities:

1,A--=0.20000

1,A-A=0.20000

1,A-C=0.20000

1,A-G=0.20000

1,A-T=0.20000

.

.

.

2,A--=0.10000

2,A-A=0.20000

2,A-C=0.25000

2,A-G=0.25000

2,A-T=0.20000

etc..

# Homework 9: output format

Coordinates of the top **10** longest contiguous state 2 segments

(Brief) annotations for the top **5** segments

Longest Segment List:

116741000 116752000

116745000 116756000

etc.. (give 10 longest from state 2)

Annotations:

Start: 116741000

End: 116752000

Overlaps with exon3 of the protein coding gene cMyc

Start: 116745000

End: 116756000

Overlaps with exon4 of the protein coding gene cMyc

# Agenda

~~Homework 9 overview~~

Connecting to a website

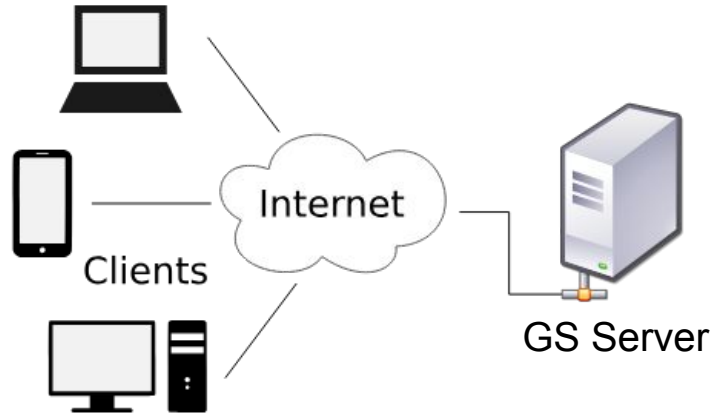
Moving data over established connections

Network vulnerabilities

Connecting to a website

# Loading a website

What actually happens when we navigate to a URL (e.g., <https://www.gs.washington.edu/index.htm>) ?



# Finding a website: DNS lookup

First, the computer has to figure out what/where  
“`www.gs.washington.edu/index.htm`” actually means

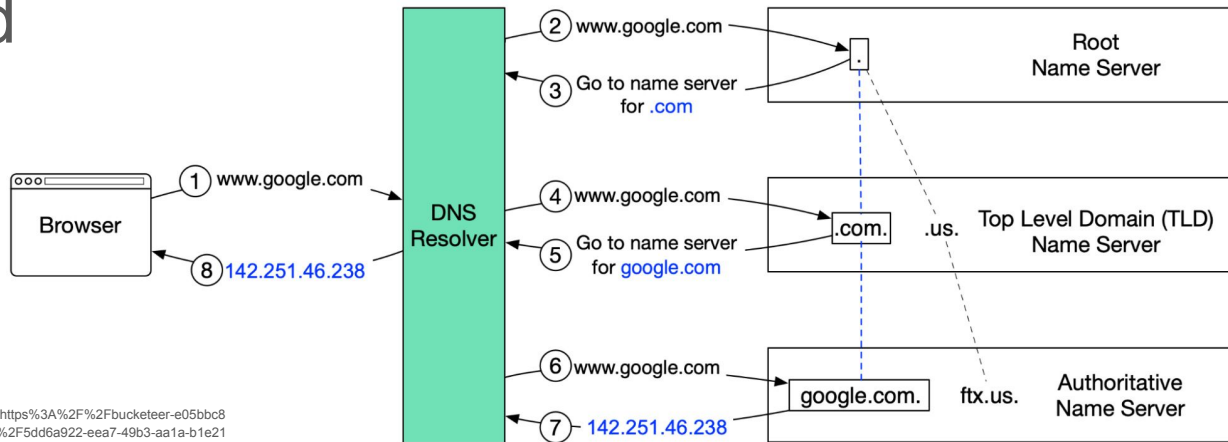
This is like using someone’s name to get their house address: converting something that’s easier for humans (our names) into something that’s easier for the system (house numbers with street names and zip codes)



# Finding a website: DNS lookup

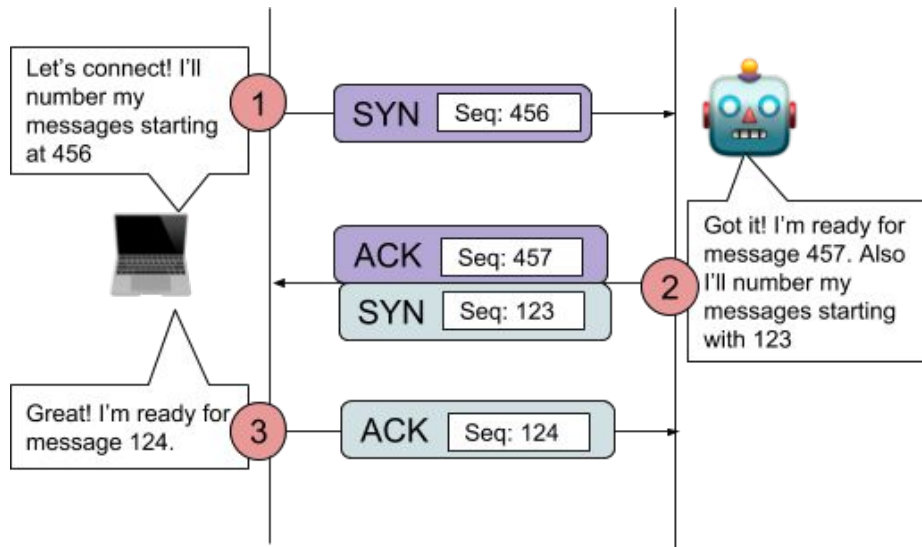
To get that address, we have to query a DNS server (provided by either your ISP or a third party like Google)

This returns an IP address, which browsers and routers can understand

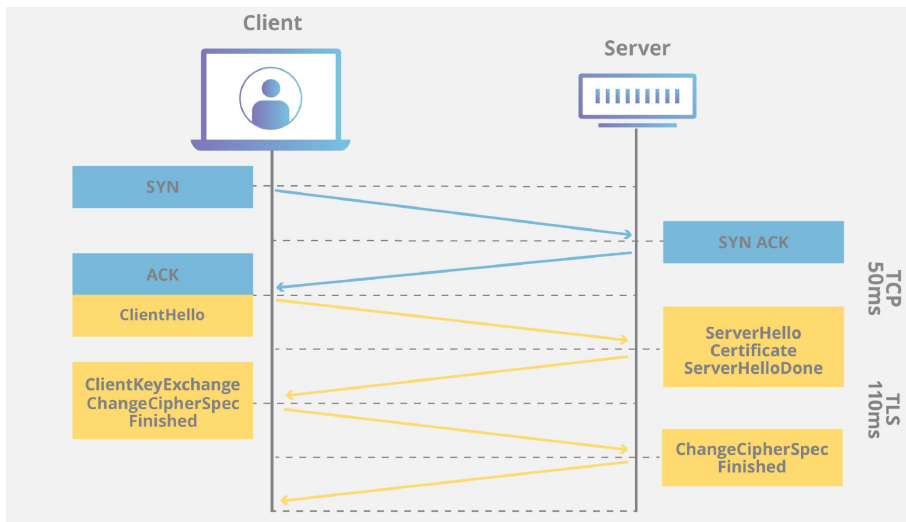


# Establishing a connection: TCP handshake

Once we know where to go, we have to go and connect to that IP address using the TCP 3-way handshake



# Establishing a connection: TLS handshake

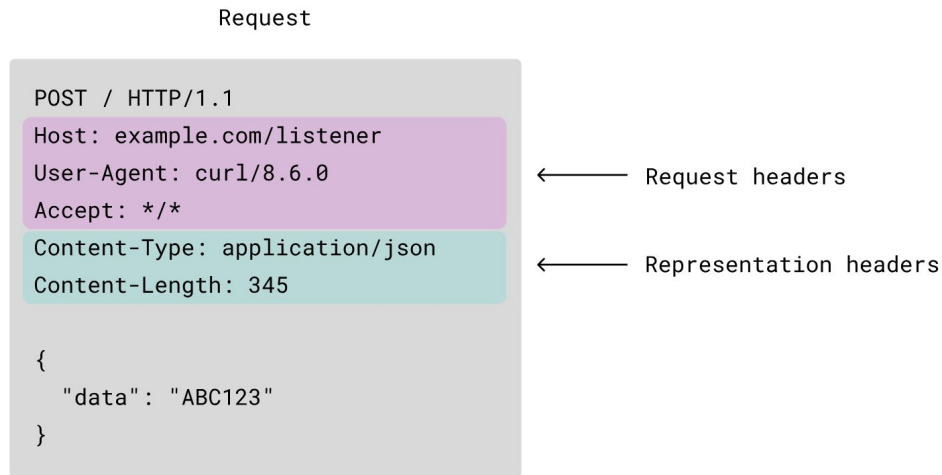


For secure connections (URLs starting with HTTPS), we also have to authenticate with the server so both client and server can encrypt and decrypt messages

Uses Diffie-Hellman!

# Loading the website: HTTP requests

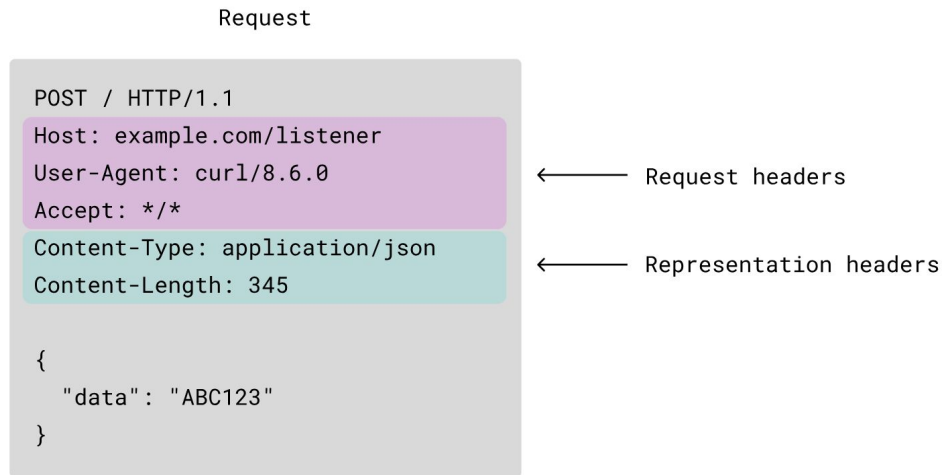
Finally, we can now send encrypted messages to the server so we can securely make HTTP requests for server resources like the actual web page we want



# Loading the website: HTTP requests

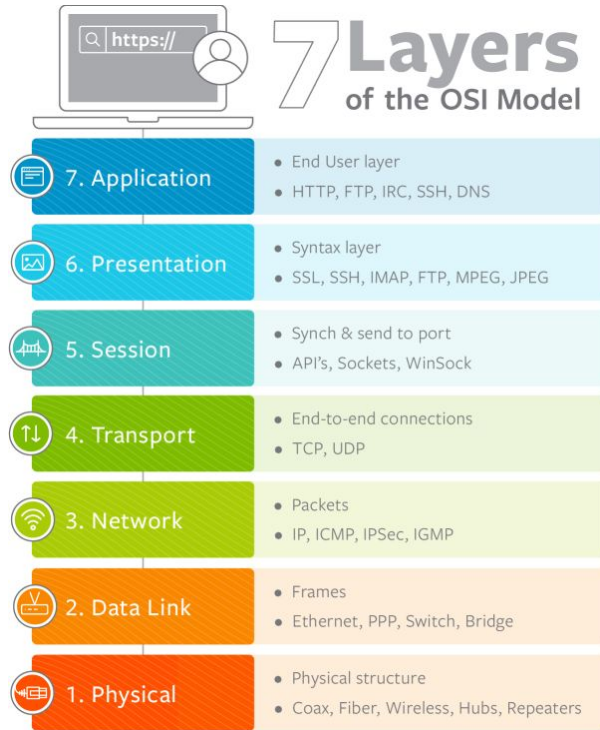
## Common HTTP requests

- GET
- POST
- PUT
- DELETE



Moving data over established connections

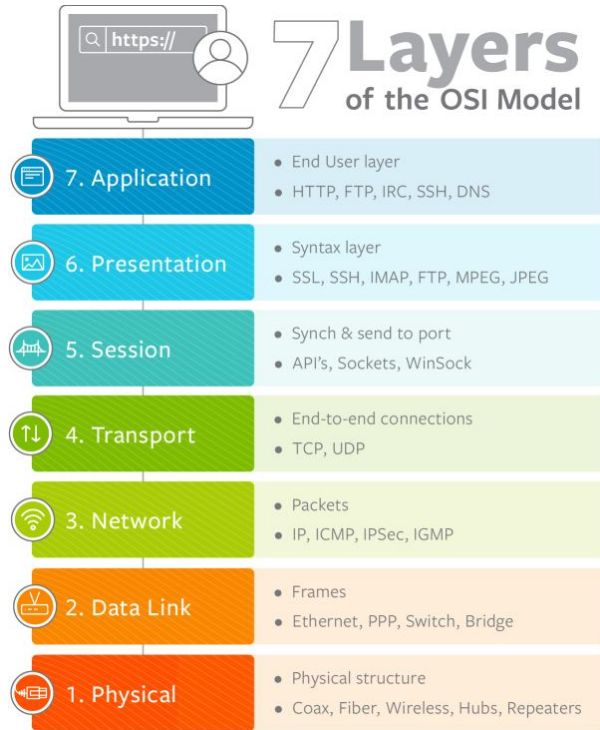
# Layers of connection



Now that we can send HTTP requests, we can send these requests down through several layers of connection

The OSI model is one such way to conceptualize these layers, but nowadays we use the TCP/IP model

# The OSI model



The Open Systems Interconnection (OSI) model describes the layers of connection needed for communication between the client and server



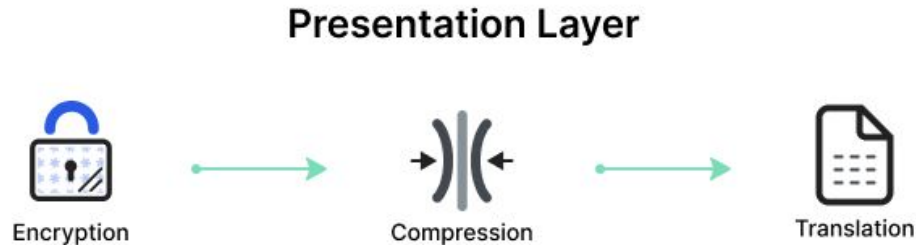
# The OSI model: application layer

Understands what the server-side web application needs from the client and encodes this information (e.g., into an **HTTP** or FTP request)



# The OSI model: presentation layer

Also known as the syntax layer; makes sure the data is formatted and compressed correctly so the application layer can understand what it has received



# The OSI model: session layer

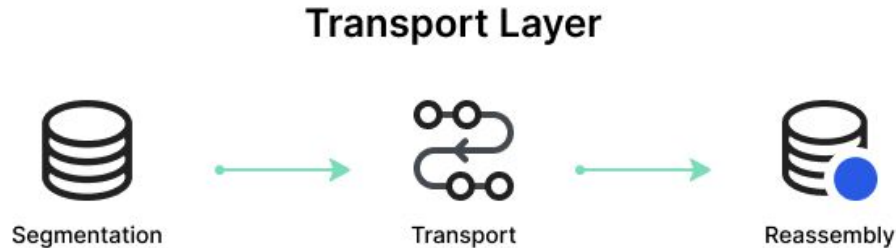
Manages and controls the connection between client and server as “sessions”



# The OSI model: transport layer

Makes sure data is transported correctly and completely by chunking up the data client-side and reassembling it server-side, fixing errors and replacing missing chunks

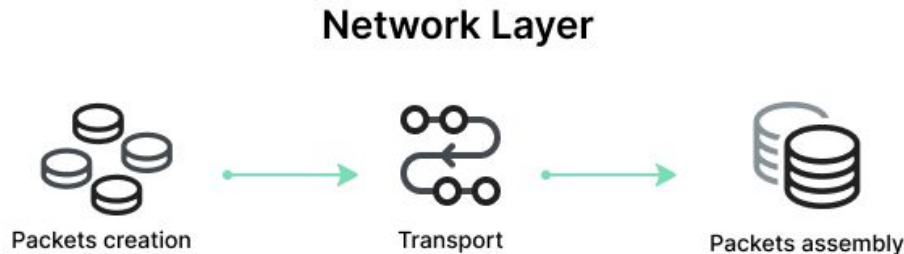
**TCP** lives here



# The OSI model: network layer

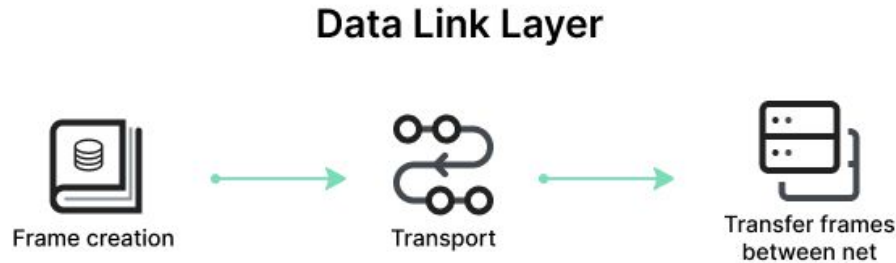
The network layer is responsible for routing packets to the right place and finding the optimal path to get there

**IP** lives here



# The OSI model: data link layer

The data link layer ensures that data is sent to the right computer on the server-side local network utilizing info like MAC addresses (unique hardware ID)



# The OSI model: physical layer

The actual physical connection between the client and the server lives inside of physical wires! Data encoded as bits are sent as electrical signals over these wires



# What does the data look like?

You can think of the actual application data as a present to be opened

The data gets iteratively wrapped in layers of information as it traverses through the OSI layers on the client side

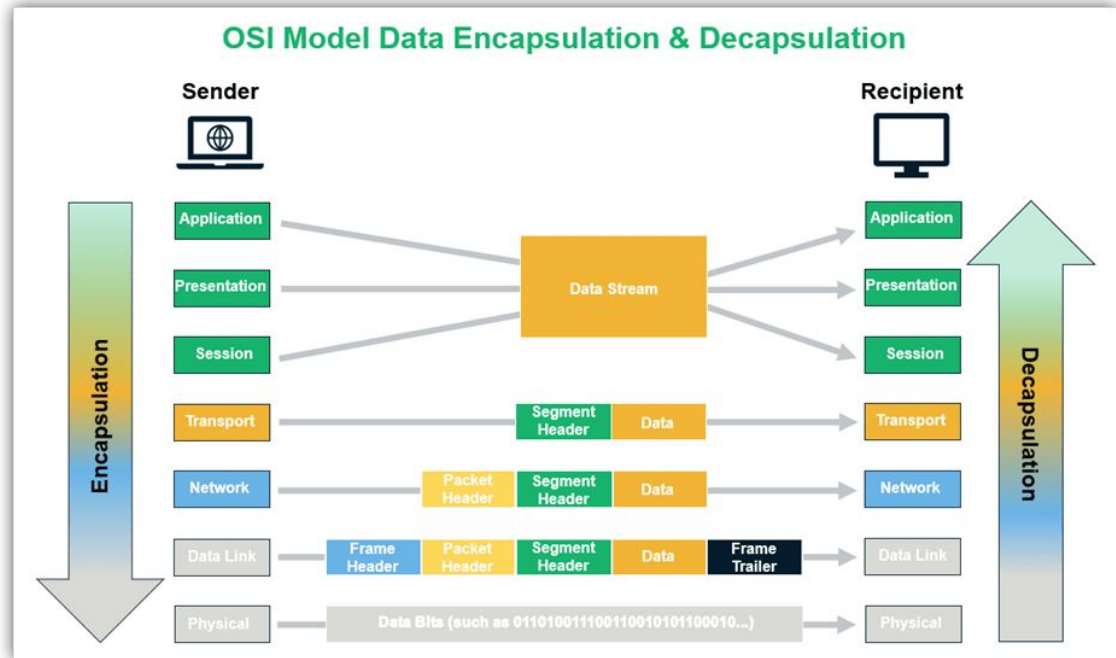
Then gets iteratively unwrapped server-side



# What does the data look like?

We call these  
wrapped data  
“packets”

Info for each layer is  
added as headers



# Network vulnerabilities

# Types of network vulnerabilities

When hosting a website or API, we must protect ourselves from outside attacks

A couple broad categories of malicious behavior:

- Injection attacks
- Denial of service
- Man in the middle

# Injection attacks

Injection attacks focus on affecting internal databases by sending data that looks valid but may cause unintended consequences

Depends on the specific implementation of the server

# Injection attacks

## Benign request:

```
POST /update-email
Content-Type: application/json

{
  "user_id": "123",
  "email": "newemail@example.com"
}
```

```
import sqlite3

def update_email(user_id, new_email):
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()

    query = f"UPDATE users SET email = '{new_email}' WHERE id = {user_id};"
    cursor.execute(query) # 🚩 Vulnerable to SQL injection!

    conn.commit()
    conn.close()
```

```
UPDATE users SET email = 'newemail@example.com' WHERE id = 123;
```

# Injection attacks

## Malicious request:

```
POST /update-email
Content-Type: application/json

{
  "user_id": "123; DROP TABLE users; --",
  "email": "hacker@example.com"
}
```

```
import sqlite3

def update_email(user_id, new_email):
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()

    query = f"UPDATE users SET email = '{new_email}' WHERE id = {user_id};"
    cursor.execute(query) # 🚩 Vulnerable to SQL injection!

    conn.commit()
    conn.close()
```

```
UPDATE users SET email = 'hacker@example.com' WHERE id = 123; DROP TABLE users; --';
```

# Injection attacks

## Malicious request handled correctly:

```
POST /update-email
Content-Type: application/json

{
  "user_id": "123; DROP TABLE users; --",
  "email": "hacker@example.com"
}
```

```
import sqlite3

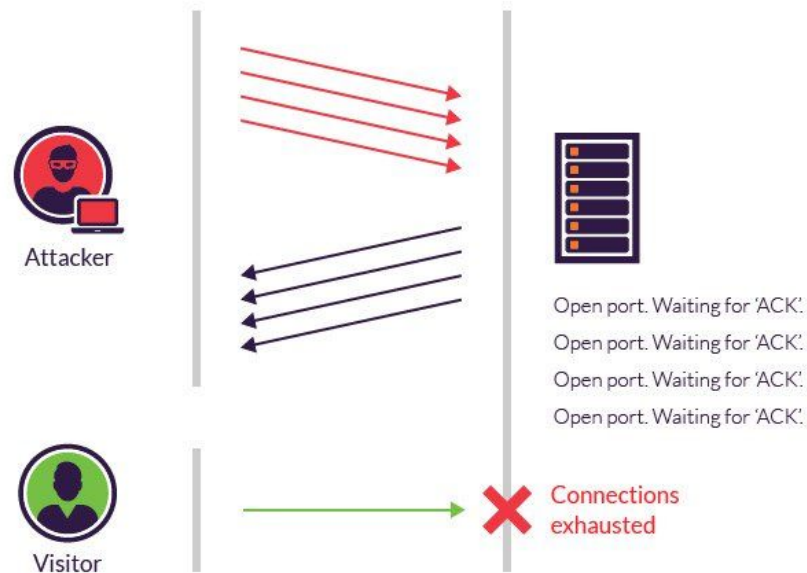
def update_email(user_id, new_email):
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()

    query = "UPDATE users SET email = ? WHERE id = ?;"
    cursor.execute(query, (new_email, user_id)) # ✅ Safe from SQL injection!

    conn.commit()
    conn.close()
```

```
UPDATE users SET email = 'hacker@example.com' WHERE id = 123;
```

# Denial of service



Denial of service attacks focus on overwhelming a system with malicious requests that prevent the server from responding to legitimate requests

One such way to do this is with a “syn flood”



# Distributed denial of service

≡ **WIRED** SECURITY POLITICS GEAR THE BIG STORY BUSINESS SCIENCE CULTURE IDEAS MERCH

LILY HAY NEWMAN

SECURITY MAR 1, 2018 11:01 AM

## GitHub Survived the Biggest DDoS Attack Ever Recorded

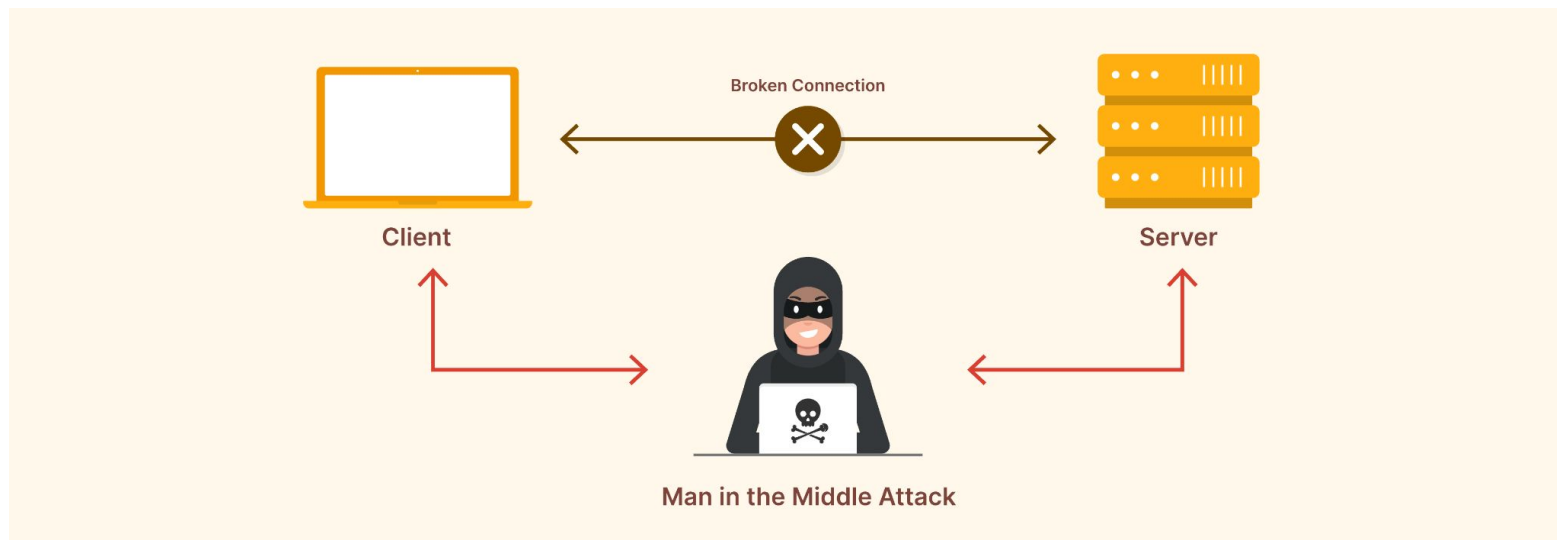
On Wednesday, a 1.3Tbps DDoS attack pummeled GitHub for 15–20 minutes. Here's how it stayed online.

A DDOS attack is a DOS attack from many parallel sources at once

Servers can employ tactics including rate limiting and the blocking of known botnet IP addresses

# Man in the middle

MITM attacks take advantage of unencrypted data transfer



# Man in the middle: example

## Connecting to a free Wi-Fi network

- The network could be set up such that all network traffic can be read directly
- So, if you log into a bank account, you might send your password in plain text (e.g., through an HTTP request) for the attacker to simply read

```
POST /login
Host: bank.example.com
Content-Type: application/json

{
  "username": "alice123",
  "password": "supersecurepassword"
}
```

# Man in the middle

The biggest protection against this is enforcing HTTPS connections

- Use the TLS handshake to exchange SSL certificate (which a well-known third party basically signs to say this is a legitimate certificate) and public/secret keys
- Now, even if traffic is spied on, it looks scrambled and can't be used when read:

```
U2FsdGVkX18nZtY5yRzDqXgXlF+DFnTwG7VqBz6vE+XhEm2Q90Kw3EjH0zh30J3+  
8G5QtZoD3TqKLV9gF+ZJ2w==
```

# Office hours

Reminder:

Homework 9 is due Sunday, March 16th at 11:59pm!