EECS 489 - FA 24
Discussion 4

DNS, Video Streaming, and Assignment 2

Outline

- Logistics
- Assignment 1 Recap
- Lecture Review: DNS & Video Streaming
- Assignment 2 Prep

Logistics

Assignment 2

Due: Friday, Oct. 11 (right before Fall Break)

- Groups of 1-3 people
- 3 AG submits per day
- 3 late days across all 3 remaining projects
- **Order of magnitude longer and more difficult than Assignment 1!**

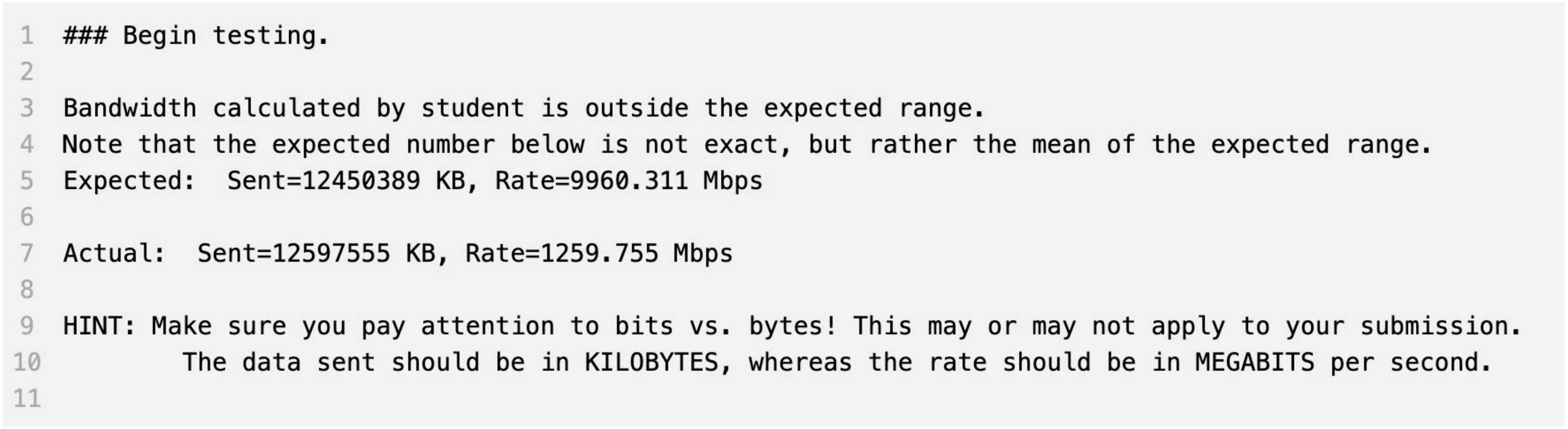
How to Develop:

- **Option 1:** Develop on AWS like Project 1; you will have to set up a VNC Client on your computer, as we need a GUI for this project.
- **Option 2:** Develop on a VM; we have created VMs for both x86 and ARM processors. These are simpler than previous semesters' VMs and should be a lot easier to use!

Assignment 1 Recap

Assignment 1: Common Mistakes

- Confusion between **bytes** and **bits**
  - Conventionally, rates are given in ___**bits** per second, while actual units of data are most common as ___**bytes**.
  - Historical Reasoning: The network doesn't care how bits are grouped together, but you need an entire byte to make sense of the data.



```
### Begin testing.

Bandwidth calculated by student is outside the expected range.
Note that the expected number below is not exact, but rather the mean of the expected range.
Expected:  Sent=12450389 KB, Rate=9960.311 Mbps

Actual:  Sent=12597555 KB, Rate=1259.755 Mbps

HINT: Make sure you pay attention to bits vs. bytes! This may or may not apply to your submission.
        The data sent should be in KILOBYTES, whereas the rate should be in MEGABITS per second.

```

Assignment 1: Common Mistakes

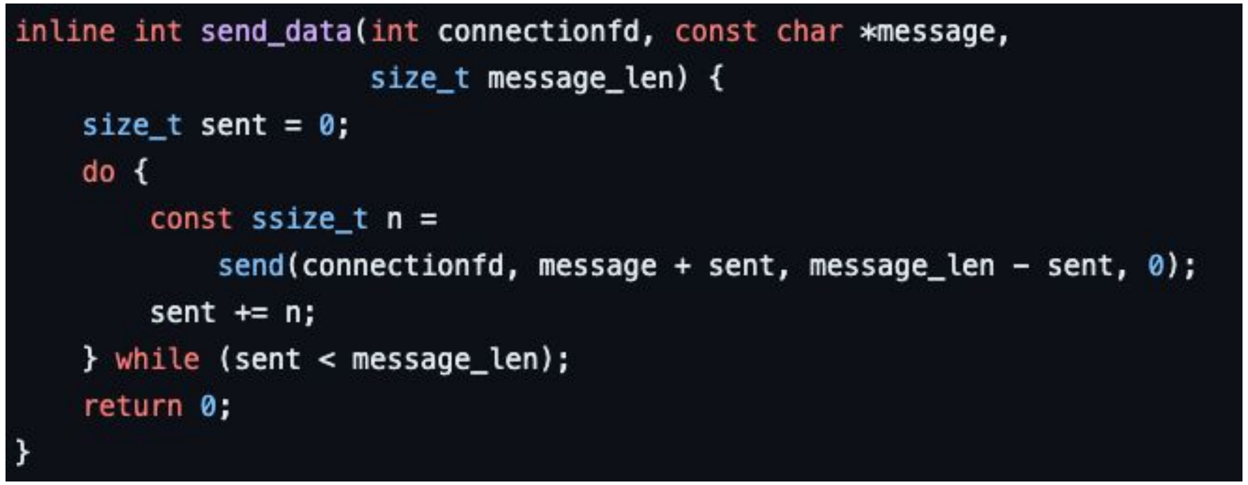
- Not calling send() in a loop!

```
ssize_t send(int sockfd, const void *buf, size_t len, int flags);
```

Sends the message described by **buf** and **len** over the connection described by sockfd.

- **Server:** `sockfd` should be the `connfd` (from accept) used to communicate with the client.
- **Client:** `sockfd` can just be the actual `sockfd`.

Returns number of bytes actually sent; this may not be all the bytes in the message. -1 on error.



```
inline int send_data(int connectionfd, const char *message,
                     size_t message_len) {
    size_t sent = 0;
    do {
        const ssize_t n =
            send(connectionfd, message + sent, message_len - sent, 0);
        sent += n;
    } while (sent < message_len);
    return 0;
}
```

Assignment 1: Common Mistakes

- Not using the MSG_WAIT_ALL flag for recv

```
ssize_t recv(int sockfd, const void * buf, size_t len, int flags);

// Example
ssize_t bytes_recvd = recv(sockfd, buffer, MSG_SIZE, 0);
// here bytes_recvd is not always == to MSG_SIZE

// But we also can do
ssize_t bytes_recvd = recv(sockfd, buffer, MSG_SIZE,
MSG_WAITALL);
// here bytes_recvd is == MSG_SIZE, but we block until then
```

Assignment 1: Common Mistakes

- Inefficient checking of FIN message
  - When building a measurement tool, we want the **network** to be the bottleneck rather than anything in our program.
  - Manually looping over every byte in each message is **inefficient** – led to people maxing out at about 2500 Mbps on the autograder rather than the ~9000 Mbps required.
  - Instead, just check the first byte of the message, ensuring that it's different from the standard data (which is all `‘\0’`s).

Assignment 1: Takeaways

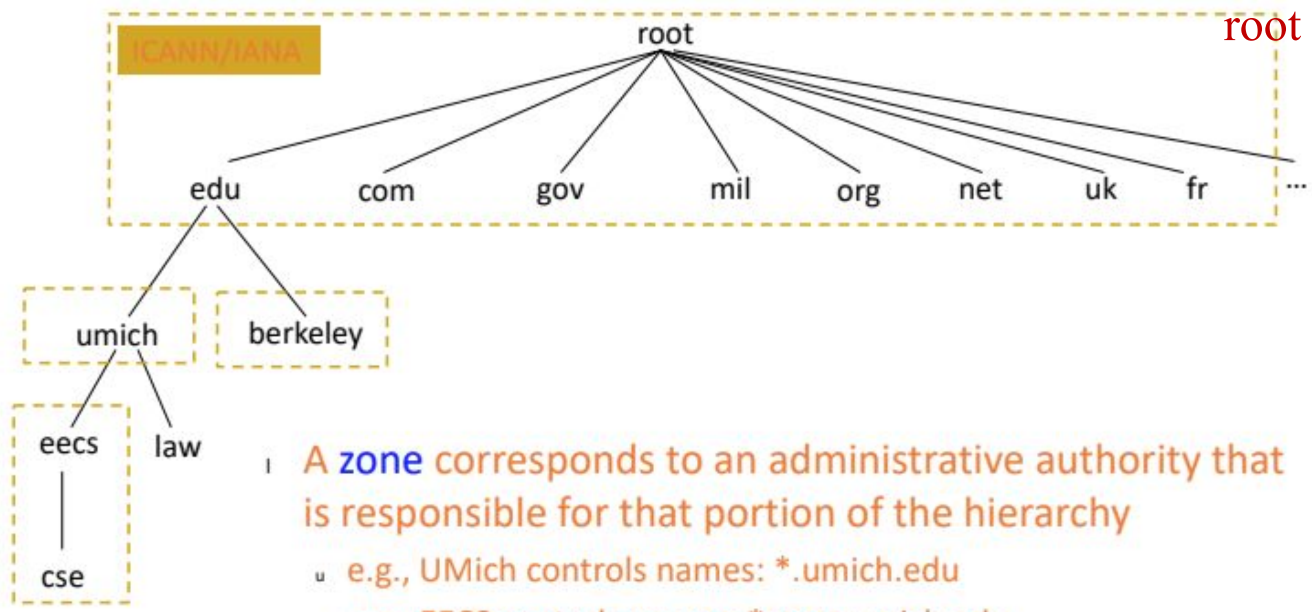
- Come to discussion! The first three bugs were discussed extensively in Weeks 1+2.
- **Read the documentation** – network functions have been around for a while, and have really clear, explicit man pages, as well as thousands of use cases online.

DNS Review

DNS: Recap

Domain Name System to map URLs (like docs.google.com) to machine addresses (like 172.217.4.46).

Hierarchical administration



root servers

top-level domain servers

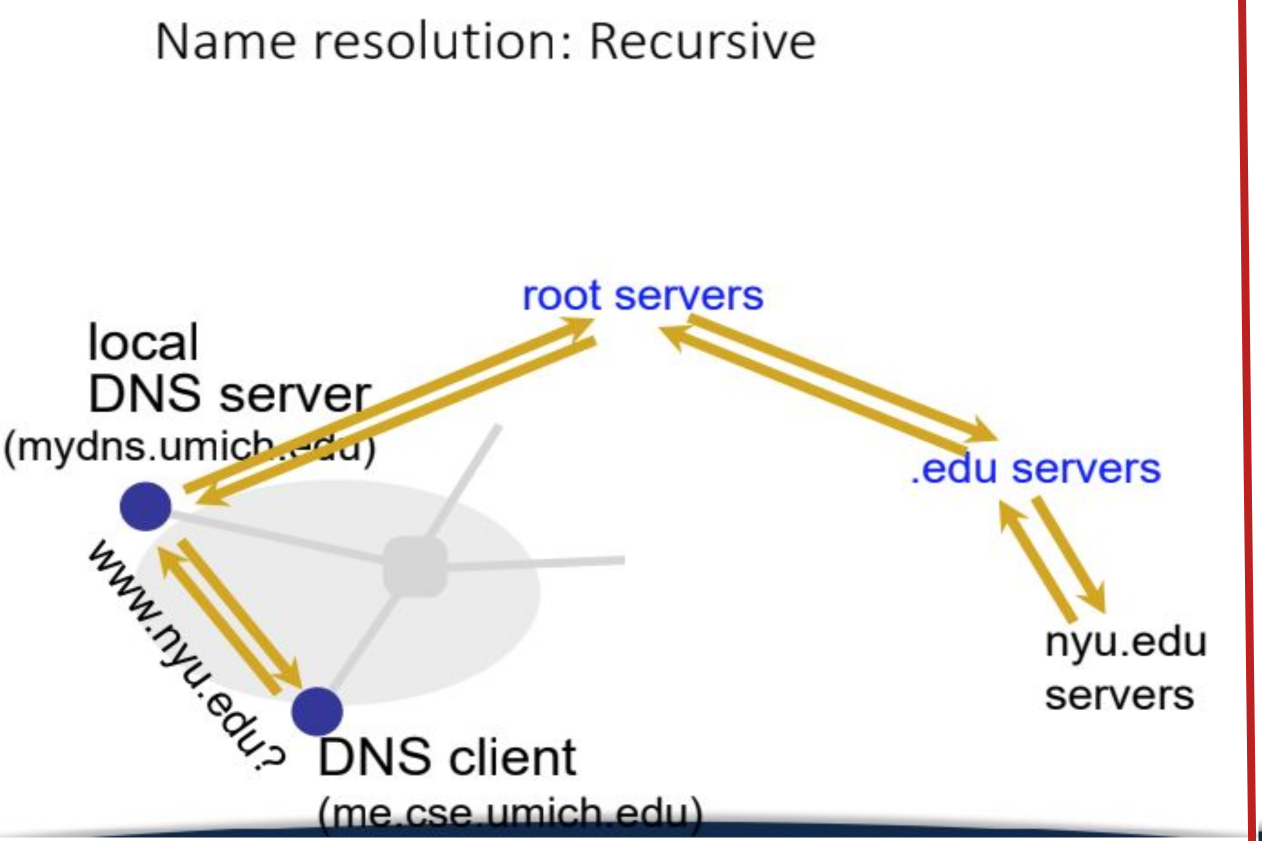
- A zone corresponds to an administrative authority that is responsible for that portion of the hierarchy
  - e.g., UMich controls names: *.umich.edu
  - e.g., EECS controls names: *.eecs.umich.edu

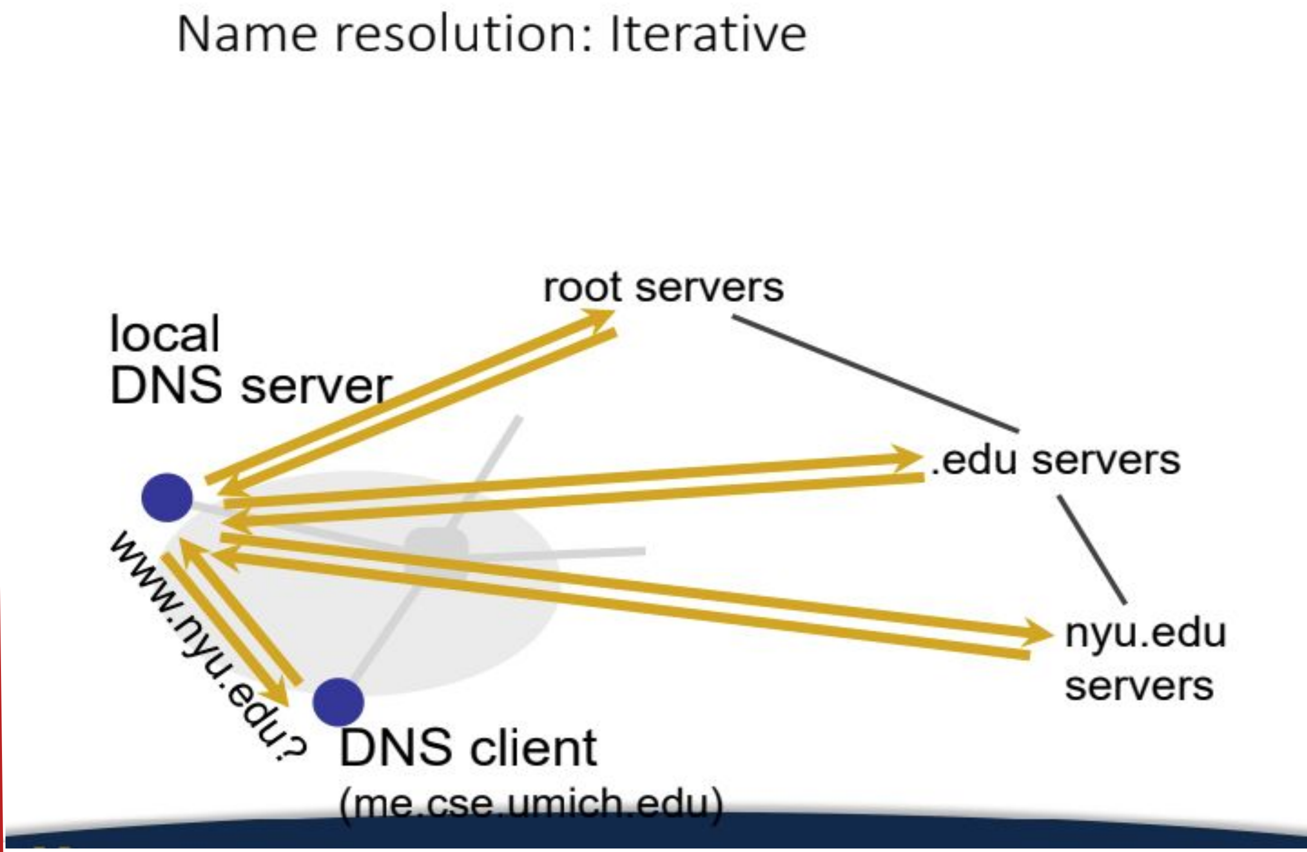
authoritative DNS servers

DNS: Protocol

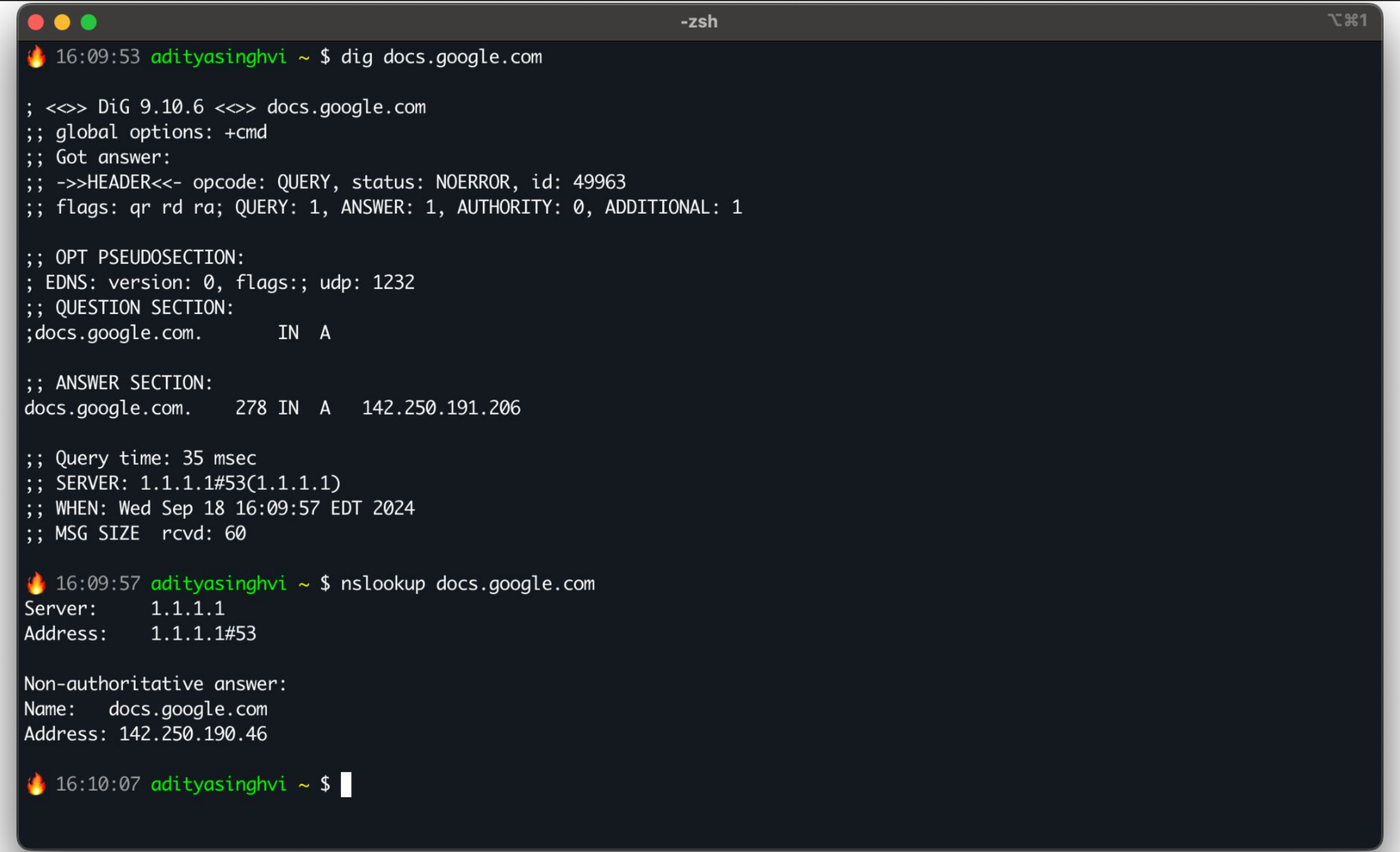
1. Host sends a **query** to local DNS server
    a. *We learn later in the semester how a host knows where to find local DNS server*
    b. Asks a question like: **What is the IP address of docs.google.com?**
2. Local DNS Server either:
    a. Has the reply cached – someone else in the network looked this up recently, so it can just respond directly.
    b. Does an iterative/recursive name resolution
        i. Might skip some steps if it has other things cached!
3. Eventually, we get a **response** from an **authoritative DNS server**, which has an A-type record mapping the hostname to the IP address
4. This response propagates back to our host, who can then remember this IP address and send IP packets to the address!

DNS: Recursive vs. Iterative





DNS: Seeing it yourself!



```
🔥 16:09:53 adityasinghvi ~ $ dig docs.google.com

; <<>> DiG 9.10.6 <<>> docs.google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49963
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;docs.google.com.       IN  A

;; ANSWER SECTION:
docs.google.com.    278 IN  A   142.250.191.206

;; Query time: 35 msec
;; SERVER: 1.1.1.1#53(1.1.1.1)
;; WHEN: Wed Sep 18 16:09:57 EDT 2024
;; MSG SIZE  rcvd: 60

🔥 16:09:57 adityasinghvi ~ $ nslookup docs.google.com
Server:     1.1.1.1
Address:    1.1.1.1#53

Non-authoritative answer:
Name:   docs.google.com
Address: 142.250.190.46

🔥 16:10:07 adityasinghvi ~ $ 
```

Lecture Based Questions - Q1

Suppose the UM EECS Department has a DNS server for all computers in the department.

How could you determine if an external web site was likely to be accessed from another computer in EECS a couple of seconds ago?

Lecture Based Questions - Q1

Suppose EECS has a DNS server for all computers in the department.

How could you determine if an external web site was likely to be accessed from another computer in EECS a couple of seconds ago?

Perform consecutive dig queries and compare the query time.

Lecture Based Questions - Q1

Suppose EECS has a DNS server for all computers in the department.

How could you determine if an external web site was likely to be accessed from another computer in EECS a couple of seconds ago?

Perform consecutive dig queries and compare the query time.

If the website was recently accessed, the first dig query would be very fast, as the address would be cached. On the other hand, subsequent queries (after a while of nobody accessing the website) might be much slower.

Lecture Based Questions - Q2

Suppose you are trying to access the page web.eecs.umich.edu/course/eecs489. You are connected to your home WiFi with its own local DNS (from your Internet Service Provider), and are not connected to the UM network. What is the order of DNS servers queried over time?

Assumptions:

- No prior caching, **recursive** name resolution
- umich.edu and eecs.umich.edu are in separate zones
- eecs.umich.edu is authoritative for all hostnames ending in .eecs.umich.edu

Lecture Based Questions - Q2

Suppose you are trying to access the page web.eecs.umich.edu/course/eecs489. You are connected to your home WiFi with its own local DNS (from your Internet Service Provider), and are not connected to the UM network. What is the order of DNS servers queried over time?

Assumptions:

- No prior caching, **recursive** name resolution
- umich.edu and eecs.umich.edu are in separate zones
- eecs.umich.edu is authoritative for all hostnames ending in .eecs.umich.edu

Lecture Based Questions - Q2

Suppose you are trying to access the page web.eecs.umich.edu/course/eecs489. You are connected to your home WiFi with its own local DNS (from your Internet Service Provider), and are not connected to the UM network. What is the order of DNS servers queried over time?

Assumptions:

- No prior caching, **recursive** name resolution
- umich.edu and eecs.umich.edu are in separate zones
- eecs.umich.edu is authoritative for all hostnames ending in .eecs.umich.edu

- **Queries:** local DNS -> root -> edu -> umich -> eecs
- **Replies:** (eecs -> umich), (umich -> edu), (edu -> root), (root -> local DNS)

Lecture Based Questions - Q3

Suppose you are trying to access the page web.eecs.umich.edu/course/eecs489. You are connected to your home WiFi with its own local DNS (from your Internet Service Provider), and are not connected to the UM network. What is the order of DNS servers queried over time?

Assumptions:

- No prior caching, **iterative** name resolution
- umich.edu and eecs.umich.edu are in separate zones
- eecs.umich.edu is authoritative for all hostnames ending in .eecs.umich.edu

Lecture Based Questions - Q3

Suppose you are trying to access the page web.eecs.umich.edu/course/eecs489. You are connected to your home WiFi with its own local DNS (from your Internet Service Provider), and are not connected to the UM network. What is the order of DNS servers queried over time?

Assumptions:

- No prior caching, **iterative** name resolution
- umich.edu and eecs.umich.edu are in separate zones
- eecs.umich.edu is authoritative for all hostnames ending in .eecs.umich.edu

Lecture Based Questions - Q3

Suppose you are trying to access the page web.eecs.umich.edu/course/eecs489. You are connected to your home WiFi with its own local DNS (from your Internet Service Provider), and are not connected to the UM network. What is the order of DNS servers queried over time?

Assumptions:

- No prior caching, **iterative** name resolution
- umich.edu and eecs.umich.edu are in separate zones
- eecs.umich.edu is authoritative for all hostnames ending in .eecs.umich.edu

- **Queries:** (local DNS -> root), (local DNS -> edu), (local DNS -> umich), (local DNS -> eecs)
- **Replies:** (root -> local), (edu -> local), (umich -> local), (eecs -> local)

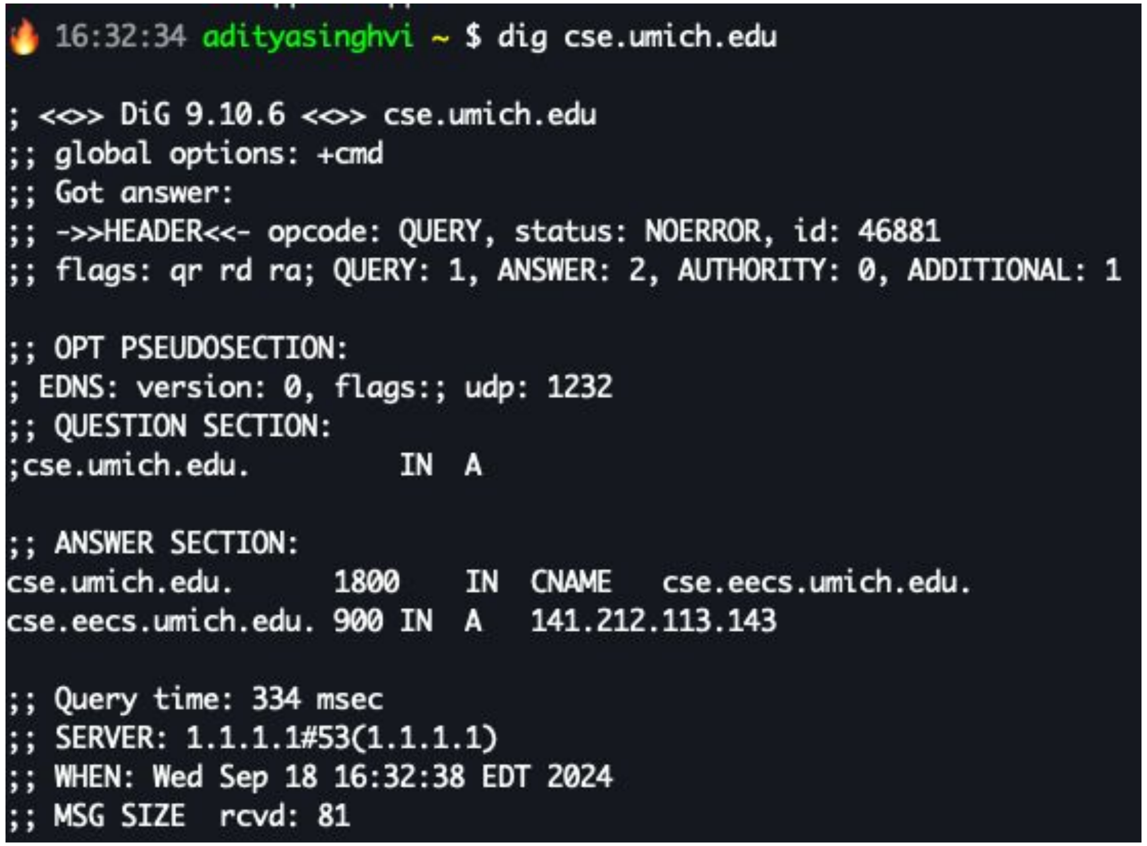
Lecture Based Questions - Q4

What is a CNAME record, and why is it useful?

Lecture Based Questions - Q4

What is a CNAME record, and why is it useful?

A CNAME (Canonical Name) record aliases one domain name to another. This allows us to point subdomains to their actual domain.



```
🔥 16:32:34 adityasinghvi ~ $ dig cse.umich.edu

; <<>> DiG 9.10.6 <<>> cse.umich.edu
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46881
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;cse.umich.edu.          IN  A

;; ANSWER SECTION:
cse.umich.edu.      1800    IN  CNAME   cse.eecs.umich.edu.
cse.eecs.umich.edu. 900 IN  A   141.212.113.143

;; Query time: 334 msec
;; SERVER: 1.1.1.1#53(1.1.1.1)
;; WHEN: Wed Sep 18 16:32:38 EDT 2024
;; MSG SIZE  rcvd: 81
```

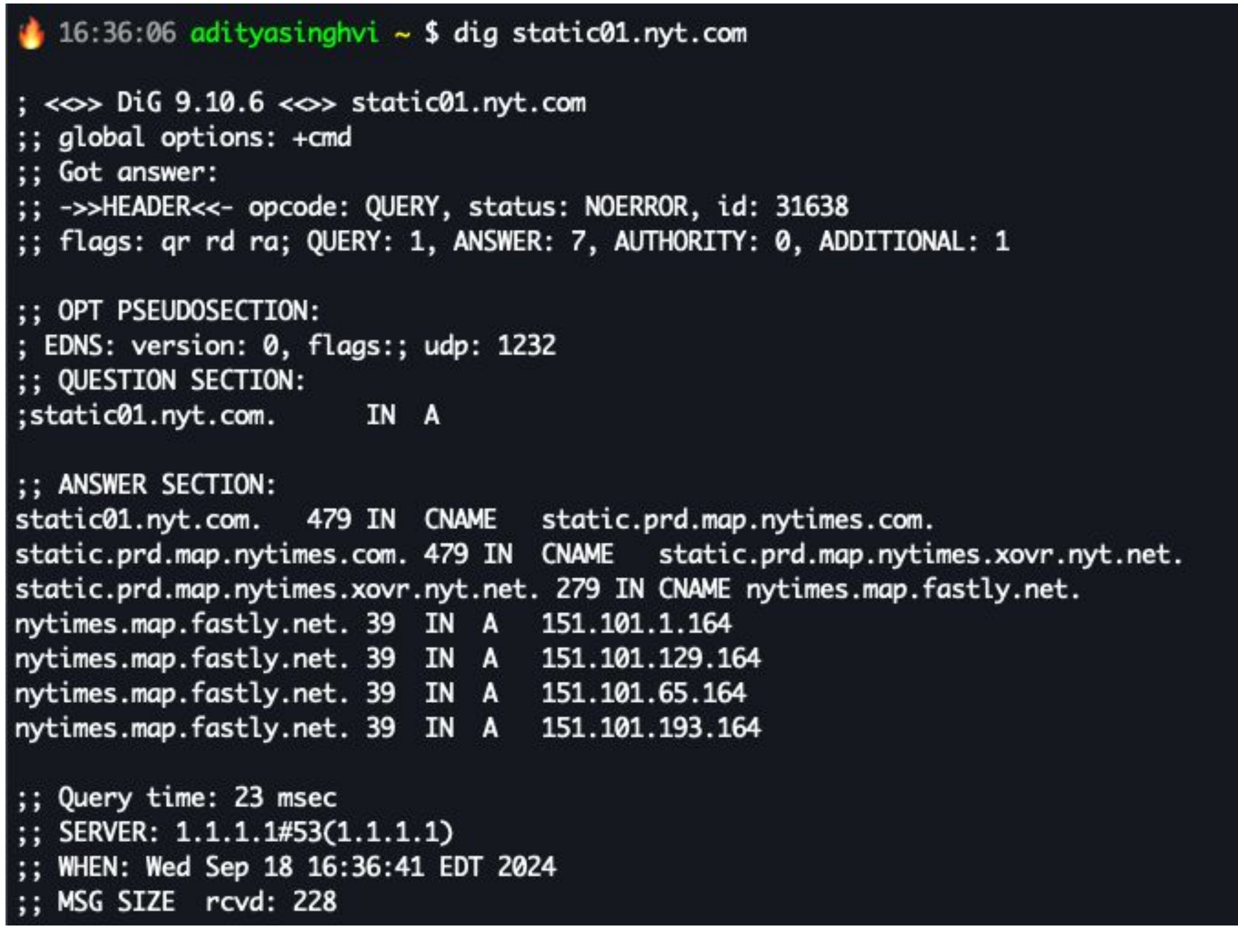
Lecture Based Questions - Q4

What is a CNAME record, and why is it useful?

A CNAME (Canonical Name) record aliases one domain name to another. This allows us to point subdomains to their actual domain.

A common use case is with CDNs:

- **Example:** Clicking on an Image URL on the NY Times website gets you something like https://static01.nyt.com/images/2024/09/18/…
- This eventually points to a fastly.net, which is a CDN! So, all requests to get images for NYT articles are actually redirected to the CDN, which can load-balance effectively.



```
🔥 16:36:06 adityasinghvi ~ $ dig static01.nyt.com

; <<>> DiG 9.10.6 <<>> static01.nyt.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31638
;; flags: qr rd ra; QUERY: 1, ANSWER: 7, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;static01.nyt.com.      IN  A

;; ANSWER SECTION:
static01.nyt.com.   479 IN  CNAME   static.prd.map.nytimes.com.
static.prd.map.nytimes.com. 479 IN  CNAME   static.prd.map.nytimes.xovr.nyt.net.
static.prd.map.nytimes.xovr.nyt.net. 279 IN CNAME nytimes.map.fastly.net.
nytimes.map.fastly.net. 39  IN  A   151.101.1.164
nytimes.map.fastly.net. 39  IN  A   151.101.129.164
nytimes.map.fastly.net. 39  IN  A   151.101.65.164
nytimes.map.fastly.net. 39  IN  A   151.101.193.164

;; Query time: 23 msec
;; SERVER: 1.1.1.1#53(1.1.1.1)
;; WHEN: Wed Sep 18 16:36:41 EDT 2024
;; MSG SIZE  rcvd: 228
```

Lecture Based Questions - Q5

What are the consequences of a malicious entity gaining control over your local DNS server?

Lecture Based Questions - Q5

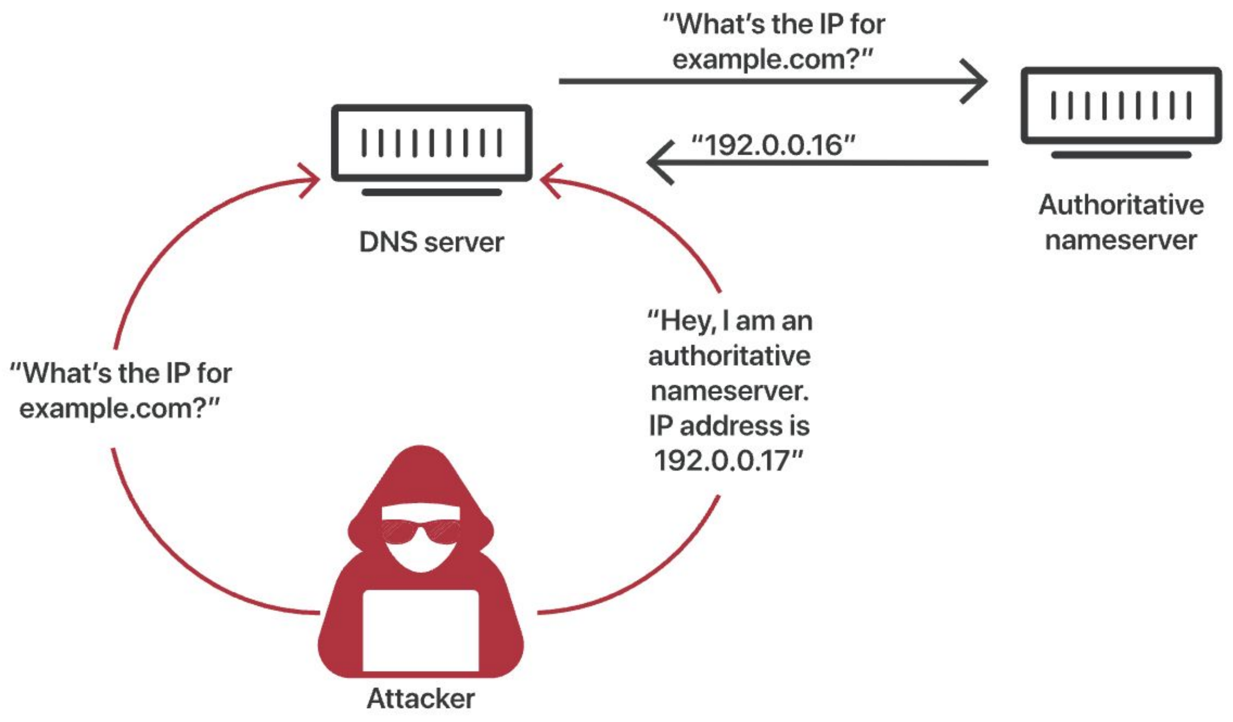
What are the consequences of a malicious entity gaining control over your local DNS server?

- Can direct you to malicious servers that resemble real servers
  - E.g. instead of accessing bankofamerica.com at its real IP address, your local DNS server tells you that it is actually a server controlled by the attacker that resembles the bank website
  - **DNS Cache Poisoning:** Attackers can make a DNS request to their local DNS server, and then spoof the response themselves before a real nameserver has a chance to respond. **This poisons the DNS cache for others using the same nameserver!**
- Can monitor your browsing (i.e. what hosts are you visiting) and block websites.

Lecture Based Questions - Q5

What are the consequences of a malicious entity gaining control over your local DNS server?

DNS Cache Poisoning Process:

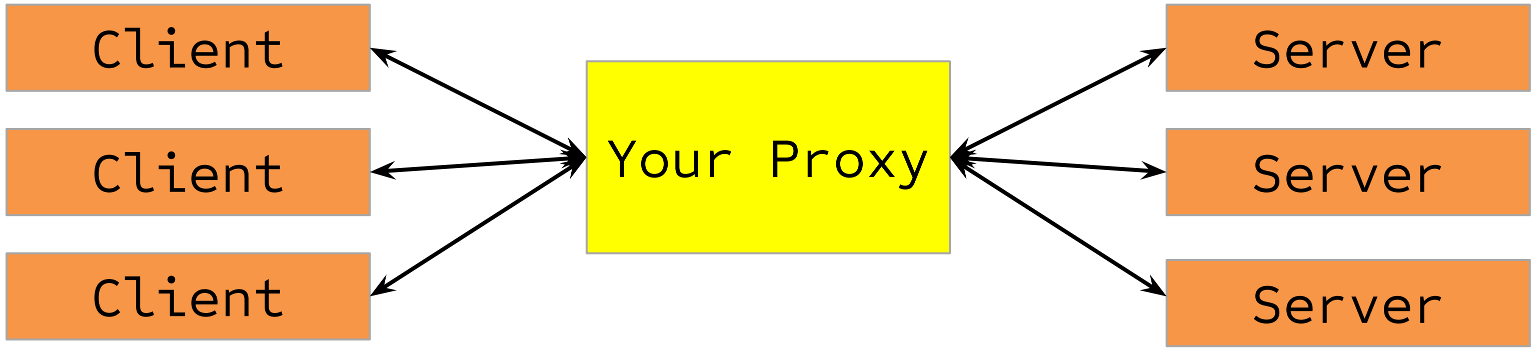

"What's the IP for example.com?"
"192.0.0.16"
DNS server
Authoritative nameserver
"What's the IP for example.com?"
"Hey, I am an authoritative nameserver. IP address is 192.0.0.17"
Attacker

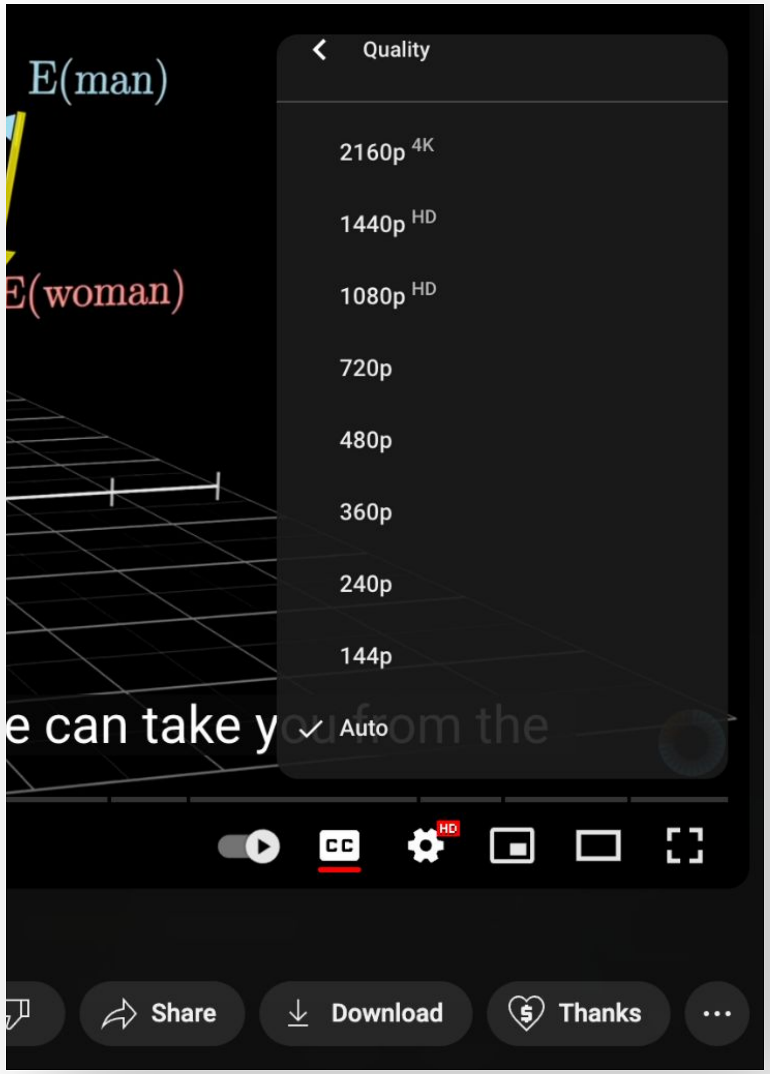
Assignment 2 Prep

Assignment 2

Part 1: HTTP Proxy for video streaming

- Sits between a client (web browser) and a standard video server.
- Modifies client requests to the video server to ask for the correct bitrate, based on constant measurement; echoes replies from the video server back to the client.
- Should be able to deal with multiple concurrent client connections!





Assignment 2

Part 2: DNS Server with Load-Balancing

- Serves requests only for the URL of the video server
- Load-balances to different IPs using two strategies (configurable when the executable is called)
  - **Strategy 1:** Round robin
  - **Strategy 2:** Geographical proximity
- No hierarchical DNS lookup – all queries go directly to this load-balancing server, which responds with an A-record

Assignment 2

Integrating Part 1 and Part 2

- Have the HTTP proxy call the DNS server when a new client connects to figure out which server to route that client’s traffic to.
- You can work on Part 1, Part 2 independently and then integrate at the end – both are testable on their own.

A2 Prep

How would you modify the P1 server to handle multiple clients simultaneously?

A2 Prep

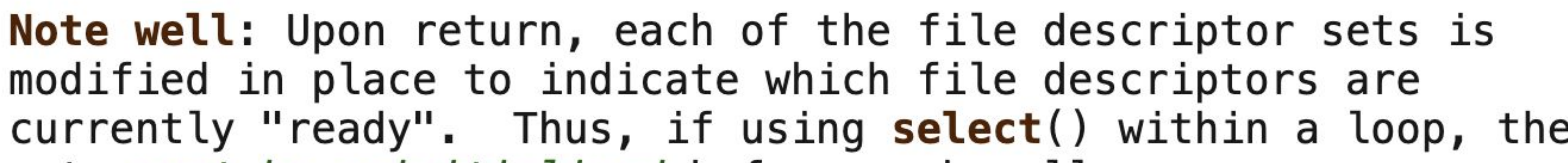
How would you modify the P1 server to handle multiple clients simultaneously?

- **Multithreading:** Create a separate process for each client and let the OS figure out how to run the threads to ensure fairness, etc.
    - Often simpler to code because you can use OS primitives for resource management
- **Polling with `select()`:** Have multiple file descriptors and continually "poll" them to see which ones need attention.
    - This can be much faster than multithreading if you have a lot of clients, as threads have a bunch of overhead
    - Can make programming more complex, and the parallelism has to be tightly integrated into the code.

A2 Prep: select()

```
#include <sys/select.h>
int select(int nfds,                    // highest # file descriptor + 1
         fd_set *readfds,               // fds to be checked for reading
         fd_set *writefds,              // fds to be checked for writing
         fd_set *exceptfds,             // fds to be checked for errors
         struct timeval *timeout);      // max wait time for anything to be ready

// For A2, we only care about readfds and have no timeout, so
// the invocation typically looks like this:
select(FD_SETSIZE, &readfds, NULL, NULL, NULL);
```



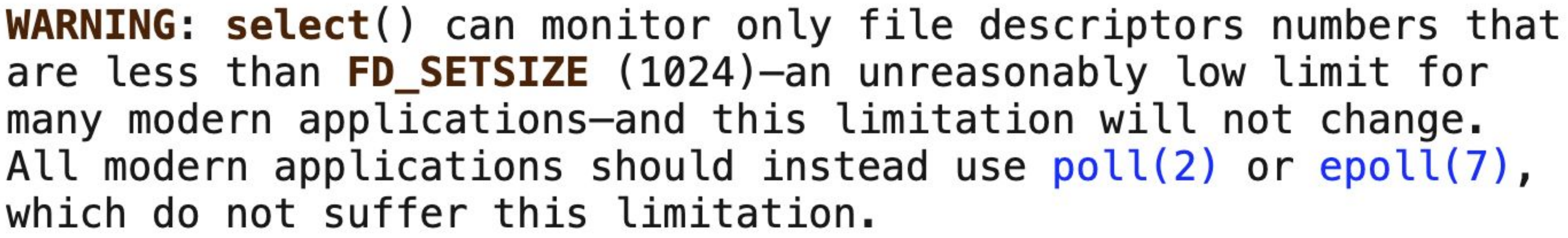
Note well: Upon return, each of the file descriptor sets is
modified in place to indicate which file descriptors are
currently "ready". Thus, if using **select**() within a loop, the

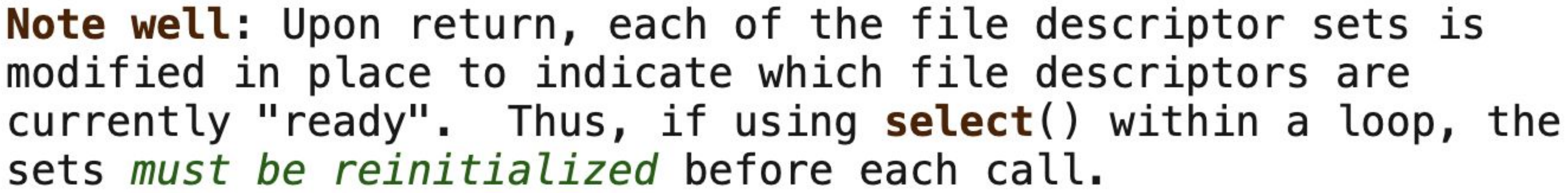
A2 Prep: select()

Some useful macros:

```
// Add fd to the set
void FD_SET(int fd, fd_set *set);
 // Remove fd from the set
void FD_CLR(int fd, fd_set *set);
// Return true if fd is in set, might not be
after select()
int FD_ISSET(int fd, fd_set *set);
// Clear all entries from set
void FD_ZERO(fd_set *set);
```

A2 Prep: select()


WARNING: select() can monitor only file descriptors numbers that are less than **FD_SETSIZE** (1024)—an unreasonably low limit for many modern applications—and this limitation will not change. All modern applications should instead use poll(2) or epoll(7), which do not suffer this limitation.


Note well: Upon return, each of the file descriptor sets is modified in place to indicate which file descriptors are currently "ready". Thus, if using **select**() within a loop, the sets *must be reinitialized* before each call.

A2 Prep: select() Demo

https://github.com/mosharaf/eecs489/tree/w24/Discussion/discussion-3-code

Credit: Zach Goldston (WN24 EECS 489 GSI)