

EECS 498 System Design of a Search Engine

Winter 2026

Lab 6: IP & Webpages

GSI: Christopher Davis
IA: Rickey Dong

Nicole Hamilton
[https://web.eecs.umich.edu/~nham/
nham@umich.edu](https://web.eecs.umich.edu/~nham/nham@umich.edu)

agenda

1. Announcements
2. Internet Protocol
3. Reading webpages

agenda

1. Announcements
2. Internet Protocol
3. Reading webpages

announcements

Reminder:

- **LinuxGetUrl** is due this Sunday (2/15)

Recommended Resources:

- Lecture 10 on Canvas
- [Beej's Guide to Network Programming](#)

agenda

1. Announcements
2. Internet Protocol
3. Reading webpages

connecting to a website

Recall from lecture:

- To request a web page, we download it from its server
- Transport/Internet Layer governs traffic between networks
 - **IP addresses** label hosts
 - To send packets to a host over the internet, we need its IP address

But how do we know the IP address for an arbitrary website?

E.g., eecs.umich.edu → IP = ???

DNS

We use the ***Domain Name System (DNS)***

- *Hierarchical* and *distributed* service to map domain names to IP addresses

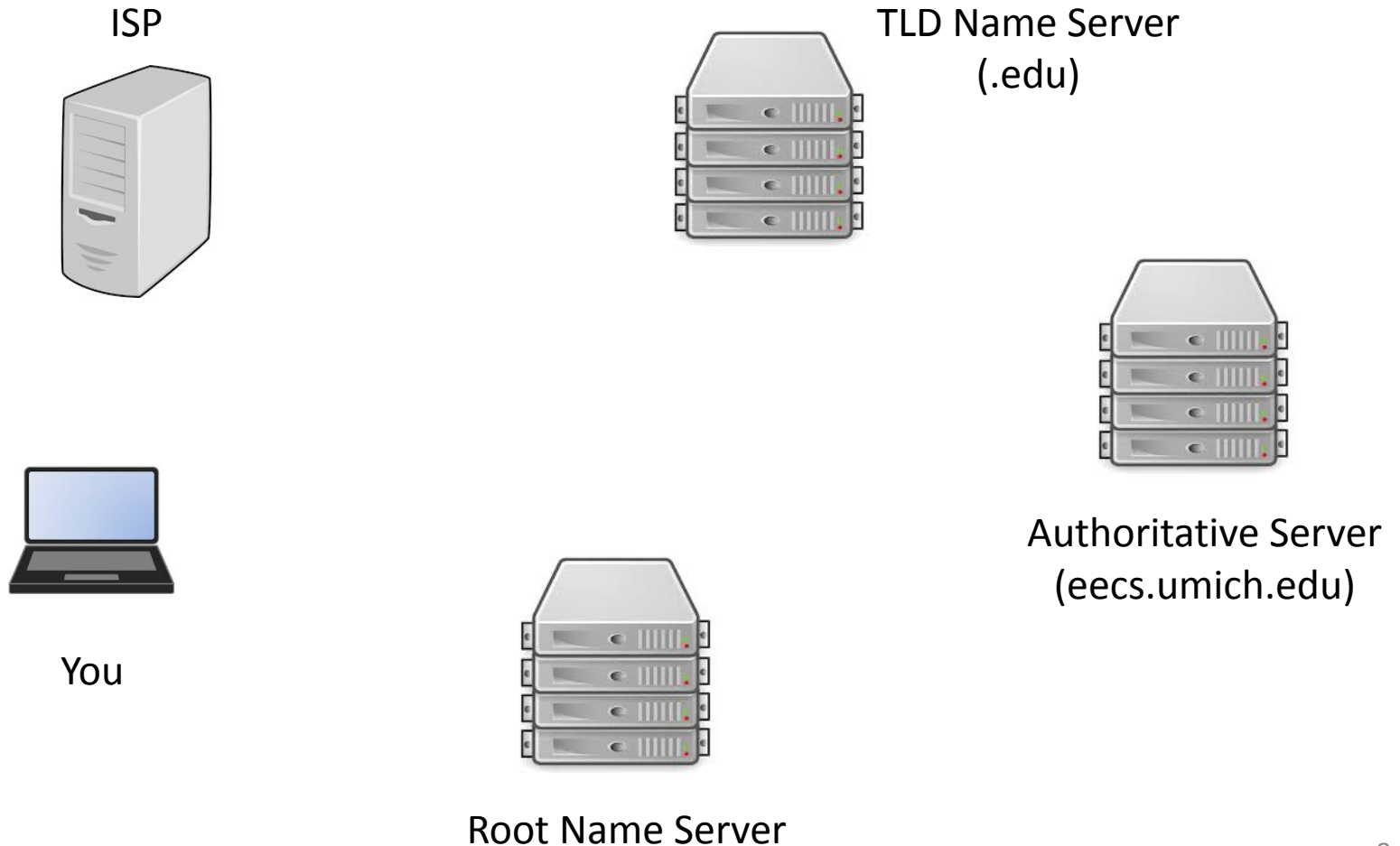
Hierarchical:

eecs.umich.edu → eecs < umich < edu

- Each level of the domain manages a name server which points to its subdomains
 - This is how domain registration works
- The root name server(s) manage records for the top-level domains (TLDs), like .com, .edu, .org, etc.

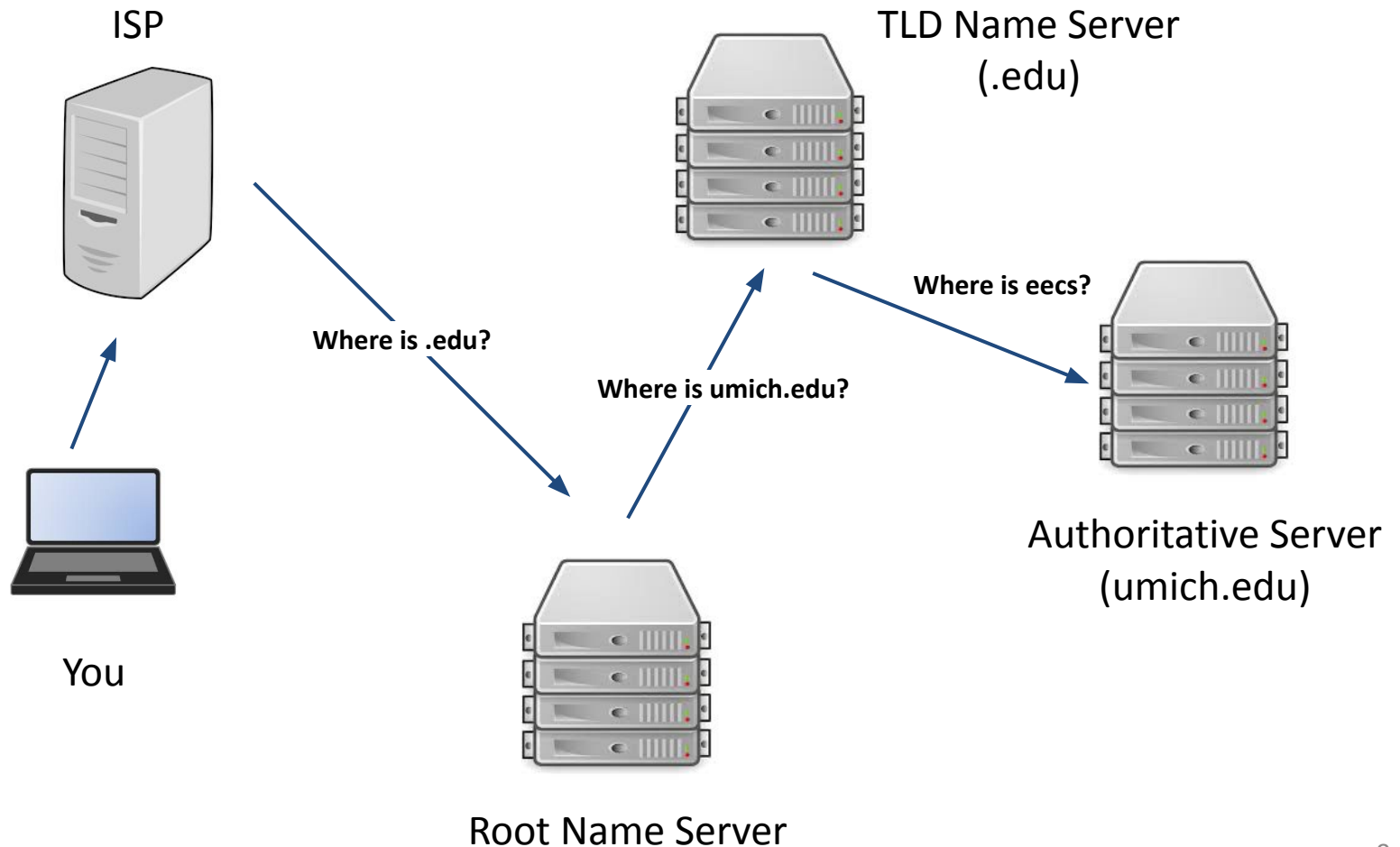
DNS example

You want to visit eecs.umich.edu on your laptop



DNS example

You want to visit eecs.umich.edu on your laptop



internet protocol (**IPv4**)

Main version was **IPv4**.

- 32-bit address
 - Represented as four 8-bit integers (0-255) separated by dots
 - E.g., 192.168.1.1
 - Capacity is $2^{32} \approx 4$ billion addresses

internet protocol (**IPv4**)

Main version was **IPv4**.

- 32-bit address
 - Represented as four 8-bit integers (0-255) separated by dots
 - E.g., 192.168.1.1
 - Capacity is $2^{32} \approx 4$ billion addresses

Problems?

internet protocol (IPv4)

Main version was **IPv4**.

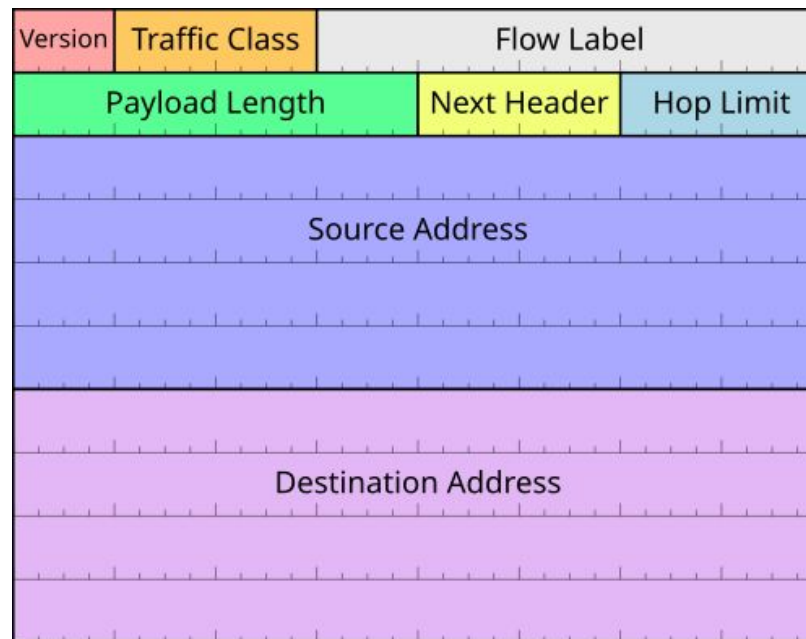
- 32-bit address
 - Represented as four 8-bit integers (0-255) separated by dots
 - E.g., 192.168.1.1
 - Capacity is $2^{32} \approx 4$ billion addresses

Problems?

- No more addresses!
 - Have to use Network Address Translation (NAT) and other hacks to get around it

internet protocol (IPv6)

- Simplified, more efficient header
 - E.g., now it's fixed-length and no checksums
- 128-bit address
 - Represented as eight groups of 4 hexadecimal numbers separated by colons
 - E.g., 2001:0db8:85a3:0000:0000:8a24:0370:7334



◀ IPv6 Header

IP coding comparison

IPv4

- Address info struct: `sockaddr_in`
- Domain/family flag: `AF_INET`

IPv6

- Address info struct: `sockaddr_in6`
- Domain/family flag: `AF_INET6`

```
struct addrinfo *address, hints;
memset( &hints, 0, sizeof( hints ) );
hints.ai_family = AF_INET; // or AF_INET6
hints.ai_socktype = SOCK_STREAM;
hints.ai_protocol = IPPROTO_TCP;
int getResult = getaddrinfo(
    "example.com", "80", &hints, &address );
int s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
connect(s, address->ai_addr, sizeof(struct sockaddr));
```

version agnostic

Problem:

Using `AF_INET/AF_INET6` explicitly hardcodes you to using a specific IP version, which your server/users may or may not support (or care about).

Solution: `AF_UNSPEC`

- Tells `getaddrinfo()` to return *any* valid addresses of host
- If DNS server has both A (IPv4) and AAAA (IPv6) records, you get both addresses back
- Allows your code to automatically and transparently handle both IP versions

handling the linked list

- `getaddrinfo()` with `AF_UNSPEC` set returns a linked list of `struct addrinfo`
 - This is a list of possible addresses for the host name
- We then iterate through and attempt to `connect()` on each of them
 - DNS might return IPv6 address first
 - If your machine or local network doesn't support IPv6, then the `connect()` call will fail
 - You must catch that failure and keep looping to the next node (e.g., an IPv4 address that you support)

```

struct addrinfo hints, *res, *p;
int status;

memset(&hints, 0, sizeof hints);
hints.ai_family = AF_UNSPEC;      // <--- IP spec agnostic!
hints.ai_socktype = SOCK_STREAM;
getaddrinfo("www.example.com", "80", &hints, &res);

// Iterate through the results list
for (p = res; p != NULL; p = p->ai_next) {
    // 1. Initialize socket using the CURRENT node's family (v4 or v6)
    int sockfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol);
    if (sockfd == -1) continue;
    // 2. Try to connect
    if (connect(sockfd, p->ai_addr, p->ai_addrlen) != -1) {
        break; // Success! We have a valid connection.
    }
    close(sockfd); // Connection failed, close and try next
}
if (p == NULL) { // We reached the end of the list without connecting
    fprintf(stderr, "Failed to connect\n");
}

```

agenda

1. Announcements
2. Internet Protocol
3. Reading webpages

Refresher on Crawler

Crawler:

1. Manage a frontier of new links to be crawled.
2. Decide what will or will not be crawled and in what order.
3. Keep track of what's already been crawled.
4. Read pages over HTTP and HTTPS.
5. Obey robots.txt files.
6. Deal with redirects.

Seed List

To start off the crawl, you need to come up with a seed list.

- cover a broad scope
- have lots of outbound links
- good quality in the first place

We've read your project plans, and we're seeing some good stuff. But there are some pitfalls you should be made aware of relating to seed lists, crawling, and reading pages...

Reddit Demo

One site we saw was Reddit. Reddit's a good choice: lots of content, outbound links, massive repository of niche information

However, crawling it may not be what you expect...

Demo:

1. Run LinuxGetSsl on https://www.reddit.com/r/uofm/comments/1nxwosu/why_does_um_have_the_cutest_chonkiest_squirrels/ (and output to a file)
2. Check the returned content. See that it has content from the post but none from the comments.
3. Run LinuxGetSsl on https://old.reddit.com/r/uofm/comments/1nxwosu/why_does_um_have_the_cutest_chonkiest_squirrels/
4. Check the returned content. See that it has content from the post AND the comments.

Server-side Rendered vs Client-side Rendered

Why did this happen?

The (new) Reddit app is a hybrid of server-side rendered (SSR) and client-side rendered (CSR). The server hosting the Reddit content sends us back an HTML page which contains JavaScript that the browser should execute to fetch the comments.

The old Reddit app is SSR, meaning that the server generates the whole HTML content and then sends it back to the client.

Our search engine vastly prefers content that is SSR. If you want to see what is equivalent to the initial HTML page that the server sends us, right click and press “View Page Source” as opposed to Inspect Element.

An Aside

If you look at the returned HTML from some pages, you might see something like this:

```
<p>That a real pic from campus? I haven't seen that big of one</p>
```

Those are HTML character entities. HTML character entities are things that purposefully replace another character because that might cause conflicts with the HTML.

Other examples include:

- & or & &
- " or " "
- ...

Could be beneficial to handle this in your HTML parser...

New York Times Demo

Another site we saw was New York Times.

Demo:

1. Run LinuxGetSsl on <https://www.nytimes.com/2026/02/11/nyregion/rats-nyc-snow-cold.html>
2. Check the returned content. Should see nothing useful.

New York Times Demo

```
<html lang="en">
<head>
  <title>nytimes.com</title>
  <style>#cmsg{animation: A
1.5s;}@keyframesA{0%{opacity:0;}99%{opacity:0;}100%{opacity:1;}}
  </style>
</head>
<body style="margin:0">
  <p id="cmsg">Please enable JS and disable any ad blocker</p>
  <script data-cfasync="false">var
dd={'rt':'c','cid':'AHrlqAAAAAMAViVUJgGhFQMAYFVsCQ==','hsh':'499AE34129FA4E4F
ABC31582C3075D','t':'fe','qp':'','s':17439,'e':'19a1e387de899d60fbccd0169e870a8456
c823bbbbfe0a5284151a87bcface987e3761f9153226dd885d95ee4e36f5df','host':'geo.
captcha-delivery.com','cookie':'FbrMklQ1YA5_M28I1LuDfrflySYYu_f1lm9TmvU4X9D5c
KpVGHQuf5N1G9ZrDco6KpxoSkmNj5LHZZCSdQl5JJcqDHPVLeiryHO4ViXlm4vIFA7j6XNx
fuUogV~r_kds'}
  </script>
  <script data-cfasync="false" src="https://ct.captcha-delivery.com/c.js">
  </script>
</body>
</html>
```

JavaScript and the Web

JavaScript is the de facto standard language for the web

All browsers understand it

JavaScript is used to interact with the Document Object Model (DOM)

Unfortunately, you can't crawl most modern web pages without a JS engine

In the Real World

How do modern search engines handle this in the real world?

- **headless browsers**

basically web browsers without a graphical user interface.

Popular ones include:

- Selenium WebDriver
- Playwright
- Puppeteer

(outside of the scope of this class...)