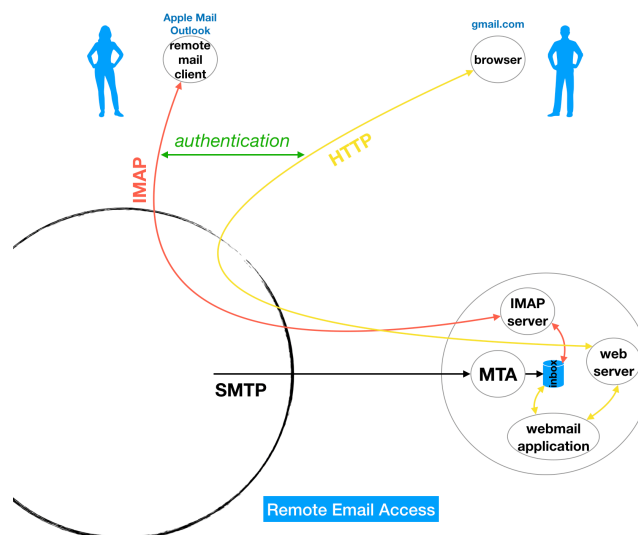


Application Layer (III)

MUA to MTA Communication

- MUA and MTA on the same host in the past
 - MUA and MTA communicate using files
 - use of host's authentication methods
- MUA and MTA communicate over a network in nowadays
 - SMTP was not designed for retrieving messages
 - it can push message to the server, but cannot pull the message from the server
 - retrieving mail: POP-3 and IMAP (include authentication)
 - POP-3 (post office protocol - version 3): port 110
 - manage messages locally, cannot access messages/folders from different machines
 - IMAP (Internet Mail Access Protocol):
 - significantly more complex
 - enable manage and organize message on the MTA, so can access folders across machines
 - obtain components of messages, useful for low bandwidth



MIME

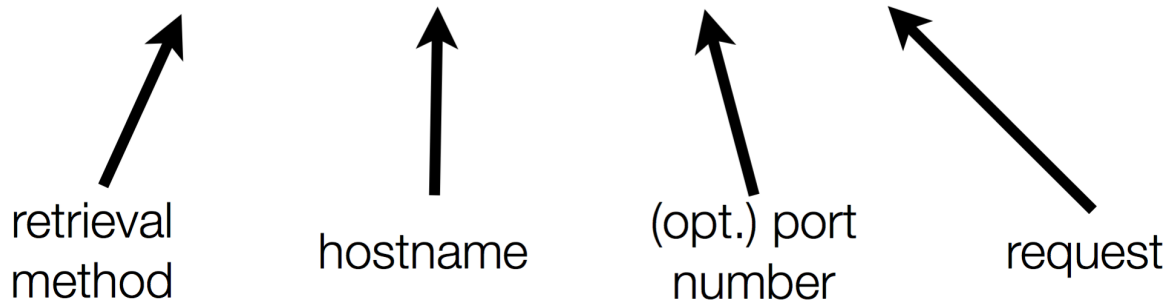
- **Problem:** SMTP was designed to deliver limited length, English text only (ASCII text)
- **Solution:** MIME (Multipurpose Internet Mail Extensions) (but can be use for HTML)
 - make everything look like text
 - pack the message (including text, characters, images, icons, audio, video, etc.) and mark it with content type so it can be unpacked and rendered on the receiving end

- Emails are **transmitted via SMTP in MIME format.**

HTTP Overview

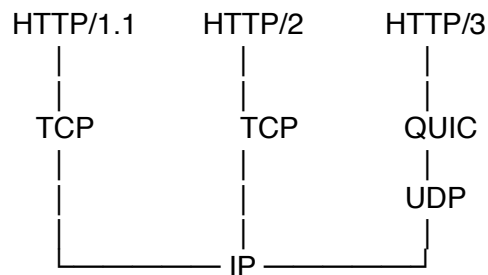
- URL (Uniform Resource Locator)

https://www.cs.colby.edu:443/dir/file.html



- Web page
 - A html file references to various objects such as images, movies, CSS, JavaScript, fonts, etc.
 - When you load one webpage, how many network requests do you think your browser actually makes? [on average 70 to over 300, html doc, external assets (CSS, JS, fonts, images, etc.), third-party services (tracking tools, analytics, ads, etc.), dynamic updates (JS makes extra background call for live data)]
 - Example: open Chrome, type colby.edu, right-click and select Inspect, go to the Network tab, and reload the page.
- Hypertext Transfer Protocol (HTTP)
 - defines how web clients request web pages from Web servers and how servers transfer Web pages to clients.
 - Client and Server architecture
 - Client side - web browser
 - Server side - web server (Apache, Microsoft Internet Information Server)
 - stateless protocol
 - servers maintain no information about the client
 - Non-persistent and persistent connections
 - Non-persistent (HTTP/1.0)
 - set up a TCP connection for each object
 - those connections may be sequential or parallel. The browser controls the degree of parallelism. (Default 5-10 parallel connections)
 - Persistent (default HTTP/1.1)
 - HTTP/1.1 persistent connection, the server leaves the TCP connection open after sending a response, and close a connection when it isn't used for a certain time (configurable). Allow back-to-back objects/requests transmission.

- HTTP/2 allows multiple requests and replies to be interleaved in the same TCP connection, and has mechanism for prioritizing HTTP message requests and replies within this connection.
- HTTP/3 runs over **QUIC** instead of TCP, and QUIC runs over **UDP**. It supports multiple independent streams. So, a lost packet affecting one stream does not necessarily block other streams. Integrates **TLS 1.3** for security. Can establish secure connections with fewer round trips. Better supports **connection migration**, e.g., a phone switching from WiFi to cellular.
- In summary,

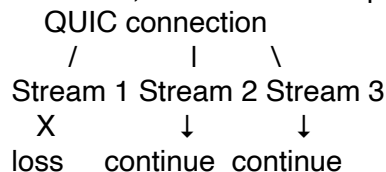


HTTP/1.1
 Persistent TCP connections
 ↓
 HTTP/2
 Multiple streams over one TCP connection
 ↓
 HTTP/3
 Multiple streams over QUIC
 → avoids TCP-level head-of-line blocking between streams

- Example:
 - Suppose a browser is downloading three resources simultaneously:
 - Stream 1: HTML
 - Stream 2: image
 - Stream 3: JavaScript

With HTTP/2, all three streams are multiplexed over the same TCP connection. If a TCP packet is lost, TCP's ordered-byte-stream requirement can temporarily prevent later received data from being delivered upwards, potentially delaying multiple HTTP streams.

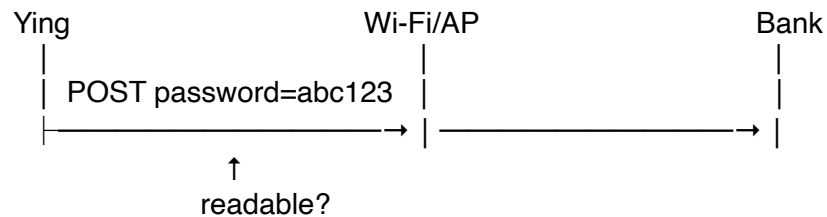
With HTTP/3/QUIC, streams are independently ordered:



A loss associated with Stream 1 does not inherently prevent data already available for Streams 2 and 3 from being delivered.

- HTTPS:

- Question: Suppose I log into my bank using HTTP. Who can see my password?



- Anyone. With ordinary HTTP, application data isn't protected by HTTP itself.
- HTTPS = HTTP communicated over a secure TLS-protected connection.
 - HTTP/1.1 and HTTP/2 encrypted by TLS before transferring over TCP, while HTTP/3's QUIC handshake incorporates TLS 1.3 or later.
- It provides:
 - confidentiality: transferred data are encrypted
 - integrity: attackers shouldn't be able to silently modify the data.
 - authentication: the browser needs evidence that it's communicating with the actual server rather than an attacker pretending to be the server.
- A server need to provide TLS certificate demonstrates it's the actual server.



• Example:

- curl -v <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html>
- curl -v <https://cs.colby.edu>
- Then with Wireshark (cs.colby.edu ip.addr == 137.146.213.6; cs.umass.edu ip.addr==128.119.245.12)

- Cookies:

- We just said HTTP is stateless. But Amazon remembers what's in my shopping cart. How?
- User-Server interaction
- most major commercial web sites use cookies
- The cookie doesn't necessarily contain the shopping cart. It can simply contain an identifier that lets the server locate state.

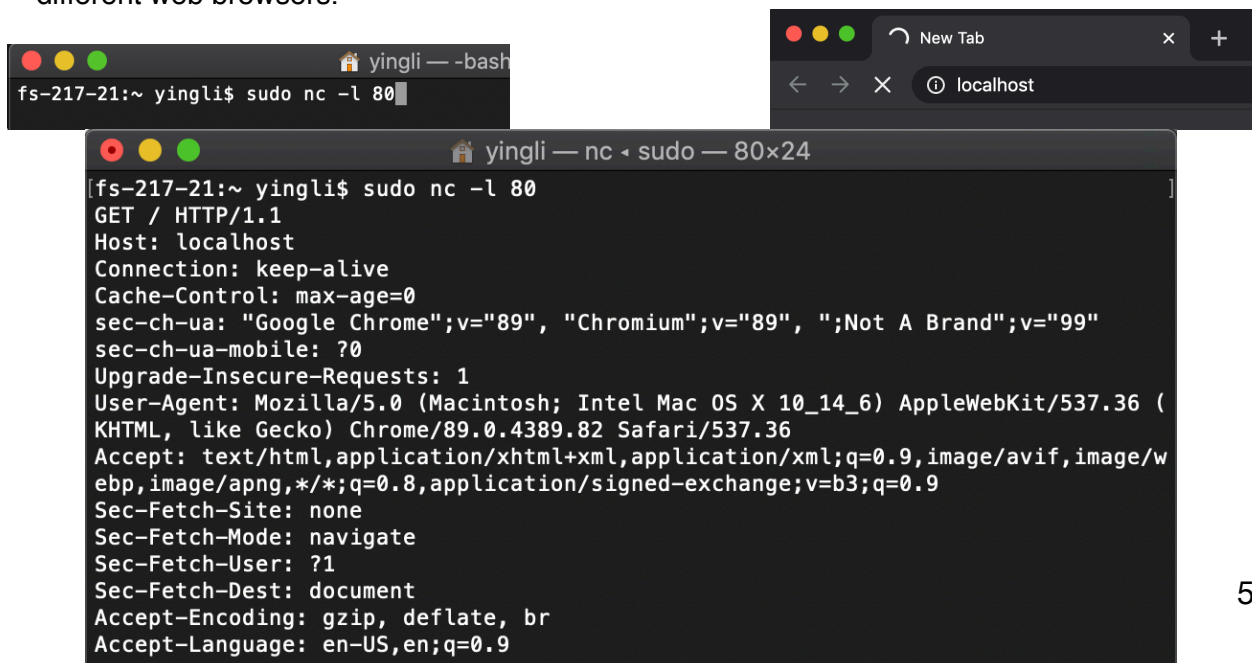
- a cookie header line in the HTTP response to let the client's browser store an identification number for the user; when the user visit the same site again, the client request contains a cookie header line in the HTTP request with the ID number; the ID number is stored by the serve in a database to keep track of the user's status (e.g., items in the Amazon cart); the browser also maintains a cookie file for the user (e.g., the ID number)
- Web caching
 - Places caching can happen:
 - proxy server: satisfies HTTP requests on behalf of an origin web server; has its own disks storage and keeps copies of recently requested objects
 - browser: users can config their computer or browsers so that all the HTTP requests direct to the proxy server first; if the proxy server doesn't have the requested object, the proxy server send a HTTP request into the cache-to-server connection; the proxy server stores a copy in its local storage and send a copy to the client browser with a HTTP response.
 - If cache misses in the browser and the proxy server, the request will be sent to the original server.
 - benefits: reduce the response time for a client request; reduce traffic on an institution's access link to the Internet (so they don't have to upgrade bandwidth quickly, therefore reduce cost); reduce traffic in the Internet as a whole, thereby improving performance for all application — Content Distribution Networks (CDNs)

HTTP GET Method

- GET
 - retrieve a representation of a resource
 - e.g., open a website

GET Examples

- Let's take a look at the details of a http request. In this example, we use localhost as a server and send a http request to itself. You may get different information in the request when using different web browsers.



The image shows a terminal window and a web browser. The terminal window, titled 'yingli — nc -l 80', displays the following output:

```
fs-217-21:~ yingli$ sudo nc -l 80
GET / HTTP/1.1
Host: localhost
Connection: keep-alive
Cache-Control: max-age=0
sec-ch-ua: "Google Chrome";v="89", "Chromium";v="89", ";Not A Brand";v="99"
sec-ch-ua-mobile: ?0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
```

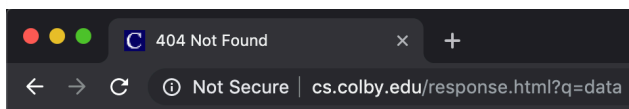
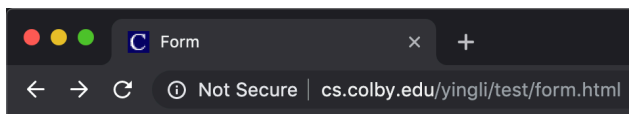
The web browser, titled 'New Tab', shows the address bar with 'localhost'.

HTTP POST Method

- POST
 - send content to a resource for resource-specific processing
 - e.g., submit an online form

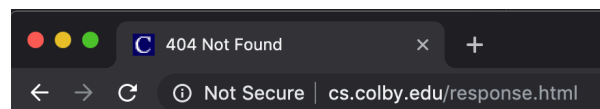
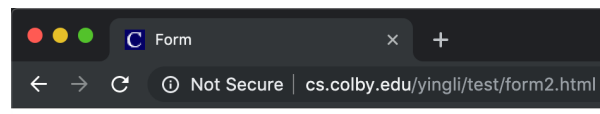
Compare POST with GET

- request `form1.html` and `form2.html` on the CS department web server from a web browser and show the difference on the URL.
 - `form1`: input data is in the URL
 - `form2`: input data is not in the URL



Not Found

The requested URL `/response.html` was not found on this server.



Not Found

The requested URL `/response.html` was not found on this server.

- show `form1.html` and `form2.html`

```

form.html
< > form.html > No Selection
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Form</title>
5 </head>
6 <body>
7   <form action="/response.html" method="GET">
8     <input type="text" name="q">
9     <input type="submit">
10  </form>
11 </body>
12 </html>

```

```

form2.html > No Selection
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Form</title>
5 </head>
6 <body>
7   <form action="/response.html" method="POST">
8     <input type="text" name="q">
9     <input type="submit">
10 </body>
11 </html>

```

- `form.html` uses GET, so the submitted data was in the URL.
- `form2.html` uses POST, so the submitted data was sent to the server implicitly (not in the URL).
- POST does not make the data secure. HTTPS does.
- But both requests couldn't get appropriate responses, as the CS department web server was not set up to response these requests.
- So how can we set up a server?