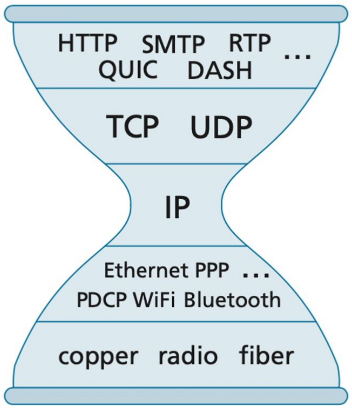
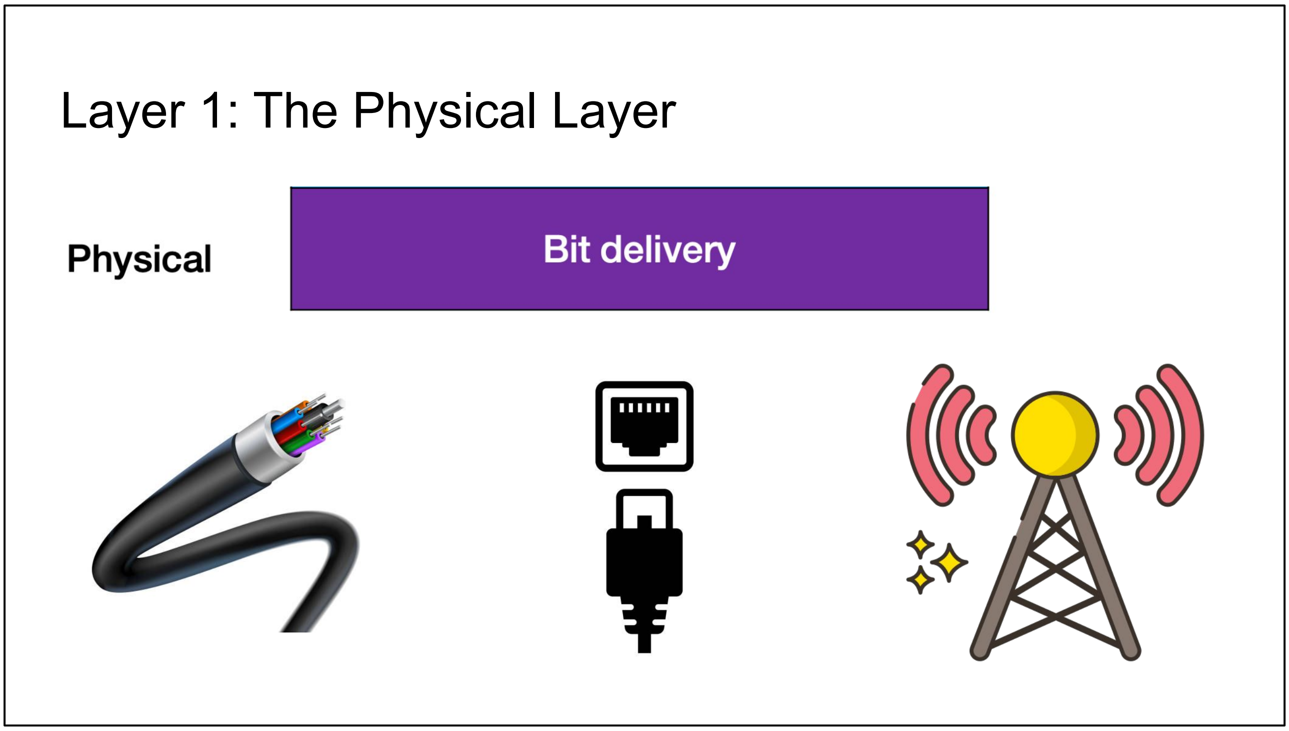
COS 316 Precept #4: Networking and HTTP

Layers of the network



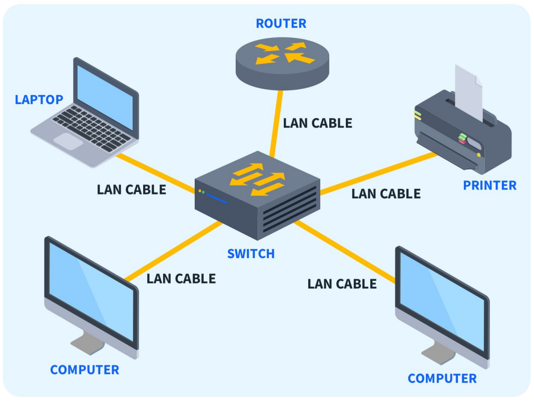
- The OSI model of the network separates it into 7-ish layers, 5 of which we'll discuss today: layers 1, 2, 3, 4, and 7
- Each of the layers builds on the layer underneath to provide additional functionality
- Each of the layers (save for IP, which we'll talk about in a second), has multiple protocols that can fit into that layer

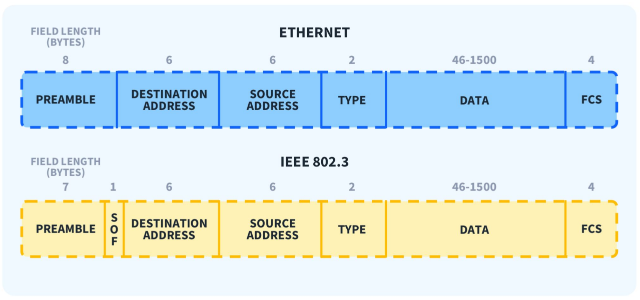


Layer 1: The Physical Layer

- Layer 1 is the physical layer
  - Layer 1 protocols describe how bits of information are sent over some communication medium
  - There's a protocol that describes how to use light to send data over fiber optic cables
  - There's another protocol that describes how to use differential signaling to send data over twisted pairs of copper wires
  - And there's another protocol that describes how to use different voltage levels to send data over coaxial cables
  - We generally don't worry very much about this layer in networking. We abstract it away.
  - But what's useful to understand about this layer is that all of the protocols within it provide a way to send bits across a communication medium, which we can refer to as bits on a proverbial wire

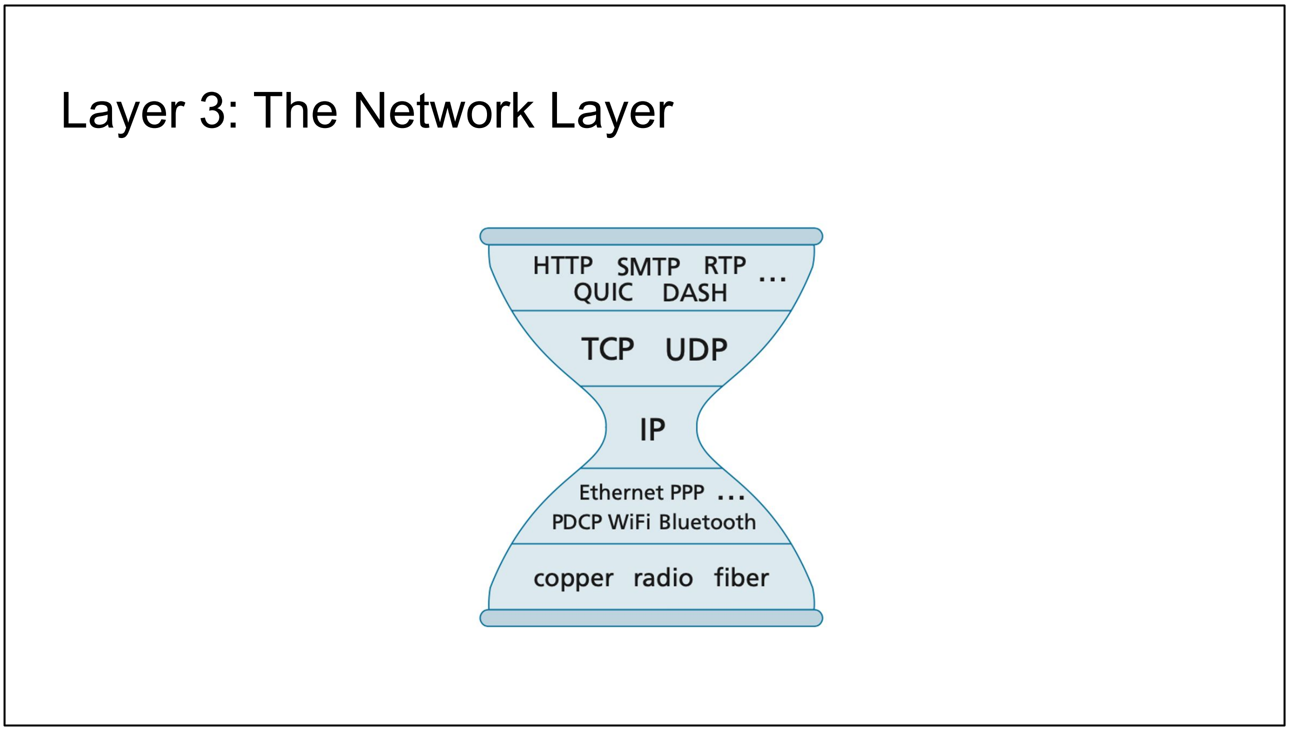
Layer 2: The Link Layer





- Now that we can get bits on a wire, we need to do something with them
- We introduce layer 2, the link layer
  - Layer 2 protocols describe how bits can be arranged into data frames and how to send those data frames between machines on the same local network
  - We've included an image of the data frame format for ethernet on the slide shown
  - This layer introduces an addressing mechanism so that machines can communicate with each other
  - Every machine that operates on a network has a layer 2 address, which is generally referred to as a MAC (media access control) address
  - These addresses are used to communicate between machines on the LOCAL network
  - For an image of what the local network looks like, take a look at the slide
  - The local network consists of everything connected to the switch at the center of the image
  - So if any of those computers wanted to send data to another one of the computers in the image, they would have to use that computer's MAC address
  - There are two things to know about MAC addresses:
    - They are static. The MAC address of each computer is hardwired into the hardware when the computer is

    - manufactured
    - They’re locally routable: they only work within the local network.
  - If a computer wants to communicate with a machine outside of its local network, then it needs to use a globally routable address, which we’ll discuss in the next section

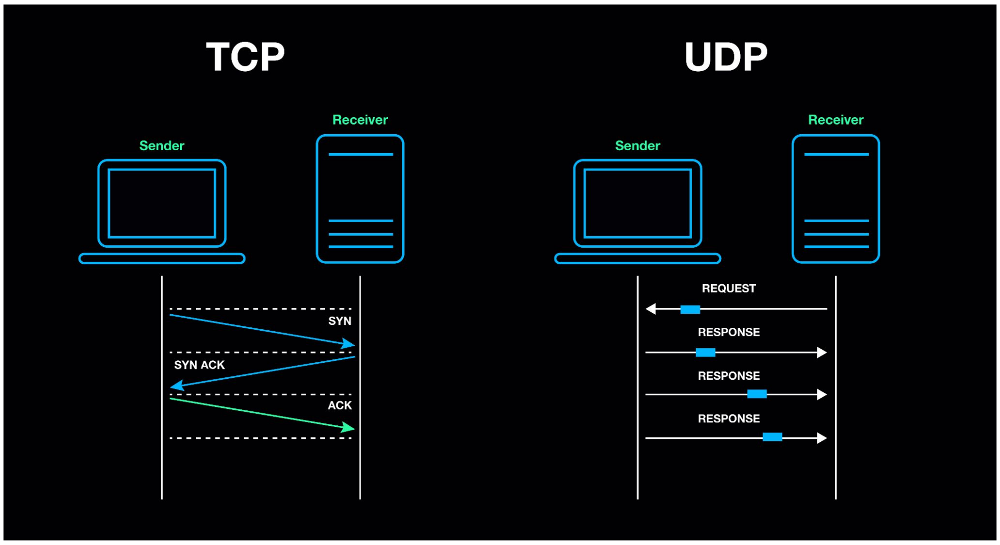


Layer 3: The Network Layer

- A shortcoming of layer 2 is that it only allows communication between machines that are on the same local network. This means that I can send and receive things to a machine that's connected to the same wifi network as my machine, but if I want to communicate with Google or some other service over the internet, I'm out of luck!
- For this purpose, we introduce layer 3, the network layer
- If you look at the network stack, you can see that there's something unique about layer 3: There's only one protocol in use!
- IP is the universally implemented protocol for machines that want to speak to other machines across the internet
- The addressing mechanism introduced at this layer is called an IP address, and it has two important attributes that differentiate it from MAC addresses
- It is globally routable and it can be assigned dynamically
- Let's refer back to MAC addresses for a second
- Remember, they're burned into a machine when the machine is manufactured
- Because of this, by their very nature, they cannot include any information about what their current location is
- But IP addresses are assigned dynamically so they can absolutely include this information
- When all of you connected your phones or computers to Princeton's WiFi network, your MAC addresses remained the same as they had always been
- But you were assigned an IP address by the WiFi network that makes it easier for other machines to find you from afar
- You can think of a MAC address as your given name and you can think of an

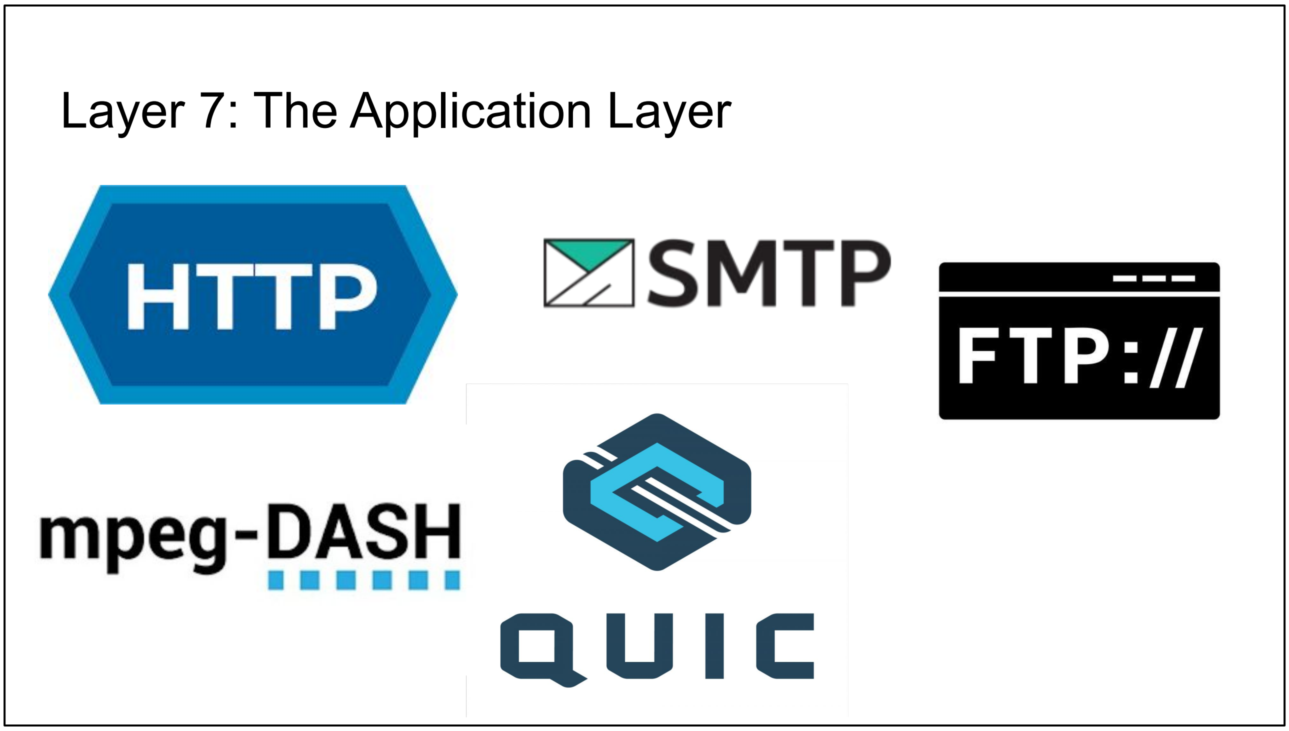
- IP address as your mailing address
- It’s easy to deliver a message to your friends because you’re near them and you know them. “Tell Chris that I said this” would work in a small area
- But that wouldn’t work if the sender lived across the country (even with assuming ‘Chris’ is globally unique)
- But they could certainly send a letter to my address and the postal service will ensure that it gets to me, because it will be delivered to my address

Layer 4: The Transport Layer



- We now have a way for machines to communicate with each other outside of their local networks
- So a machine in California can send data to a machine in New York, and the internet will perform the steps to deliver that data
- Except, it won't always do that
- There are two important things that are missing from the network layer:
    - The first is ordered, reliable delivery. The internet will ATTEMPT to deliver sent data to its intended recipient and in the correct order, but there's no built-in mechanism to ensure that happens
        - If some device in the network goes down temporarily and drops the packet, neither the sender or intended recipient will know about it, and the network will do nothing to resend the dropped packet
        - In addition, if packets are reordered for some reason (weirder things have happened), how will the recipient know?
    - The second thing that's missing from the network layer is the ability to multiplex IP addresses
        - When you connected to Princeton's Wifi network, your machine was given an IP address
        - This is what your machine will use to communicate with the outside world
        - But there are many applications running on the machine and you want your machine to be able to differentiate the traffic that's meant for Spotify vs Chrome

        - So your machine needs the ability to multiplex its internet connection
- There are two primary transport protocols in use today
- We touched on them in a previous precept:
    - TCP and UDP
- TCP addresses both points from earlier. It provides ordered, reliable delivery on top of the network layer, in addition to allowing applications on a machine to multiplex a single network connection
- UDP only addresses the latter point. It doesn't provide any kind of ordering or delivery guarantees, but it allows for multiplexing,



Layer 7: The Application Layer

HTTP

SMTP

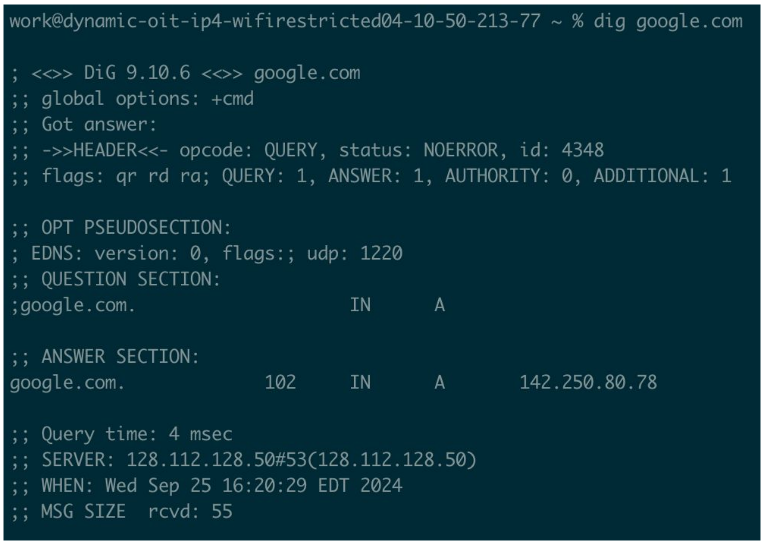
FTP://

mpeg-DASH

QUIC

- The application layer consists of protocols that need use these multiplexed connections
- A few examples are SMTP, which is used for exchanging emails, DASH, which is used for streaming video, and HTTP, which is used for the exchange of web pages

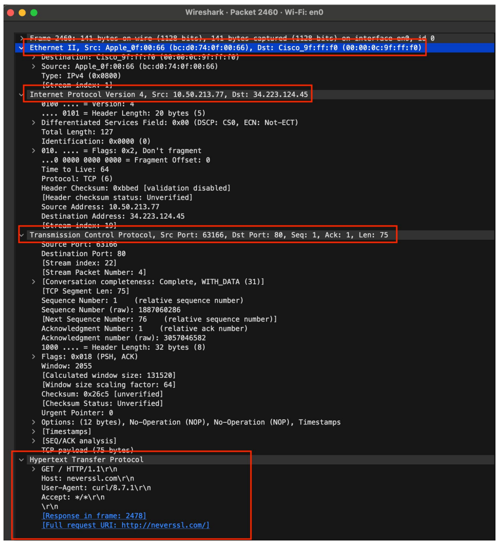
DNS



```
work@dynamic-oit-ip4-wifirestricted04-10-50-213-77 ~ % dig google.com

; <<>> DiG 9.10.6 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4348
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1220
;; QUESTION SECTION:
;google.com.                    IN      A

;; ANSWER SECTION:
google.com.             102     IN      A       142.250.80.78

;; Query time: 4 msec
;; SERVER: 128.112.128.50#53(128.112.128.50)
;; WHEN: Wed Sep 25 16:20:29 EDT 2024
;; MSG SIZE  rcvd: 55
```

- A quick note on DNS
- We know that every machine can reach every other internet-connected machine using their IP addresses
- But IP addresses aren't human readable and it would be difficult to remember the IP address for the servers that hold all of the websites that we want to use
- DNS is an optimization that's meant to make it easier for humans to specify another machine's address
- DNS provides a service that maps hostnames to IP addresses
- When you type a website into your browser, the DNS service is used under the hood to translate that name to an IP address
- We've included a screenshot of dig, a tool that lets us see this translation
  - Try this with Google.com or something like neverssl.com
  - Don't try this with Princeton.edu. There's some certificate magic that stops it from working

The layers in practice

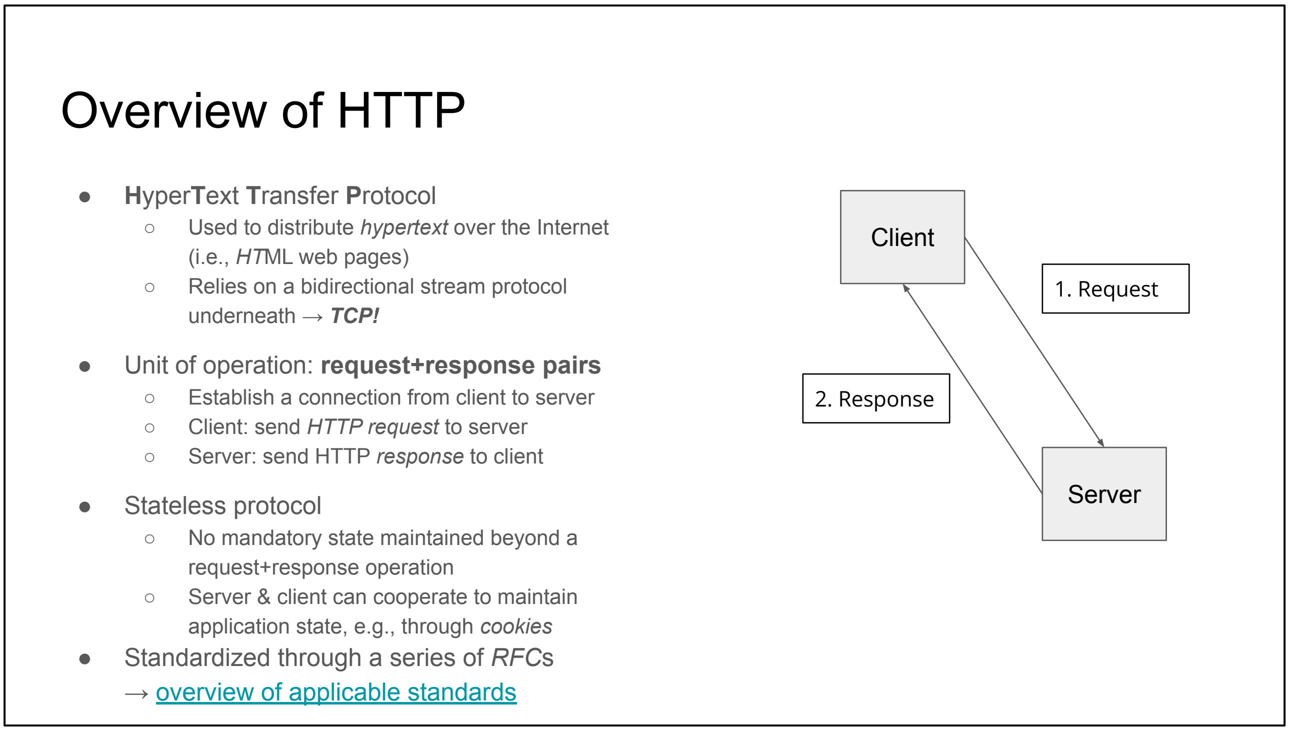


Wireshark · Packet 2460 · Wi-Fi: en0

```
> Frame 2460: 141 bytes on wire (1128 bits), 141 bytes captured (1128 bits) on interface en0, id 0
v Ethernet II, Src: Apple_0f:00:66 (bc:d0:74:0f:00:66), Dst: Cisco_9f:ff:f0 (00:00:0c:9f:ff:f0)
    > Destination: Cisco_9f:ff:f0 (00:00:0c:9f:ff:f0)
    > Source: Apple_0f:00:66 (bc:d0:74:0f:00:66)
      Type: IPv4 (0x0800)
      [Stream index: 1]
v Internet Protocol Version 4, Src: 10.50.213.77, Dst: 34.223.124.45
      0100 .... = Version: 4
      .... 0101 = Header Length: 20 bytes (5)
    > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      Total Length: 127
      Identification: 0x0000 (0)
    > 010. .... = Flags: 0x2, Don't fragment
      ...0 0000 0000 0000 = Fragment Offset: 0
      Time to Live: 64
      Protocol: TCP (6)
      Header Checksum: 0xbbed [validation disabled]
      [Header checksum status: Unverified]
      Source Address: 10.50.213.77
      Destination Address: 34.223.124.45
      [Stream index: 19]
v Transmission Control Protocol, Src Port: 63166, Dst Port: 80, Seq: 1, Ack: 1, Len: 75
      Source Port: 63166
      Destination Port: 80
      [Stream index: 22]
      [Stream Packet Number: 4]
    > [Conversation completeness: Complete, WITH_DATA (31)]
      [TCP Segment Len: 75]
      Sequence Number: 1    (relative sequence number)
      Sequence Number (raw): 1887060286
      [Next Sequence Number: 76    (relative sequence number)]
      Acknowledgment Number: 1    (relative ack number)
      Acknowledgment number (raw): 3057046582
      1000 .... = Header Length: 32 bytes (8)
    > Flags: 0x018 (PSH, ACK)
      Window: 2055
      [Calculated window size: 131520]
      [Window size scaling factor: 64]
      Checksum: 0x26c5 [unverified]
      [Checksum Status: Unverified]
      Urgent Pointer: 0
    > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    > [Timestamps]
    > [SEQ/ACK analysis]
      TCP payload (75 bytes)
v Hypertext Transfer Protocol
    > GET / HTTP/1.1\r\n
      Host: neverssl.com\r\n
      User-Agent: curl/8.7.1\r\n
      Accept: */*\r\n
      \r\n
      [Response in frame: 2478]
      [Full request URI: http://neverssl.com/]
```

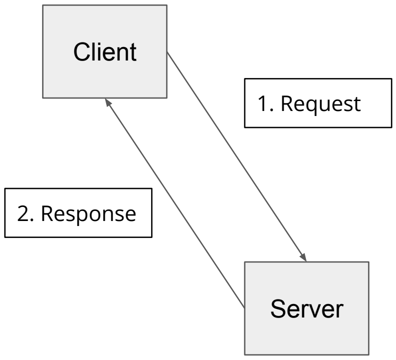
- Let’s see how this works in practice:
- This is a capture of a single packet
- I issued the command “curl -v neverssl.com” in order to request a web page from my terminal.
- While doing this, I recorded the network traffic from my machine
- You can see the header for each layer of the network stack
- At the top is the layer 2 header with associated MAC addresses
- Below that is the layer 3 header
- Below that is the layer 4 header
- And below that is the layer 7 header

Questions?



Overview of HTTP

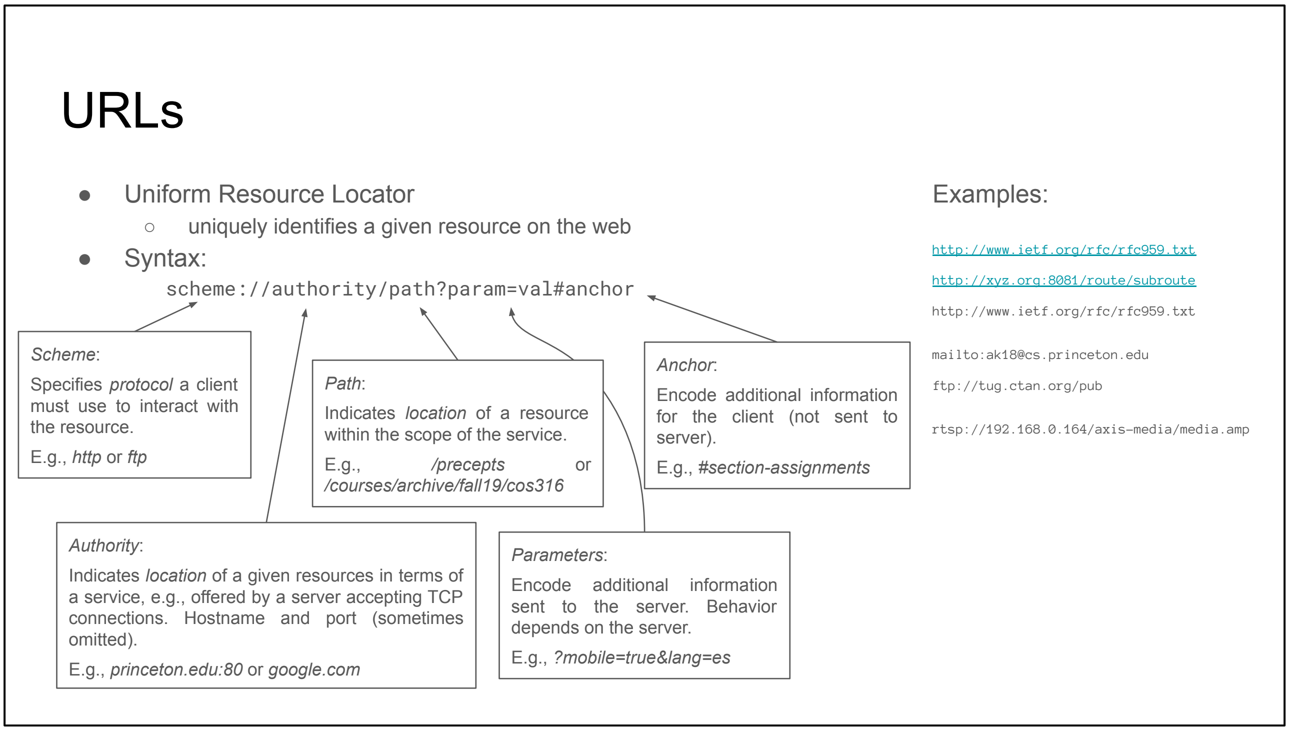
- **H**yper**T**ext **T**ransfer **P**rotocol
  - Used to distribute *hypertext* over the Internet (i.e., *HT*ML web pages)
  - Relies on a bidirectional stream protocol underneath → ***TCP!***
- Unit of operation: **request+response pairs**
  - Establish a connection from client to server
  - Client: send *HTTP request* to server
  - Server: send HTTP *response* to client
- Stateless protocol
  - No mandatory state maintained beyond a request+response operation
  - Server & client can cooperate to maintain application state, e.g., through *cookies*
- Standardized through a series of *RFC*s → overview of applicable standards



- Let's talk about HTTP!
- HTTP is an application layer protocol for distributing hypertext, in other words, HTML web pages
- It is short for HyperText Transfer Protocol and it is built on top of TCP, the transport layer protocol

- HTTP operates in a call and response fashion
- Once a client has established a connection, it can issue an HTTP request, and the server will issue an ensuing response

- It is a stateless protocol, meaning that there's no state required to be kept by the client or server in order for the protocol to work
- However, there is an optional way for state to persist: Cookies!



URLs

- Uniform Resource Locator
  - uniquely identifies a given resource on the web
- Syntax:

```
scheme://authority/path?param=val#anchor
```

Scheme:
Specifies *protocol* a client must use to interact with the resource.
E.g., *http* or *ftp*

Authority:
Indicates *location* of a given resources in terms of a service, e.g., offered by a server accepting TCP connections. Hostname and port (sometimes omitted).
E.g., *princeton.edu:80* or *google.com*

Path:
Indicates *location* of a resource within the scope of the service.
E.g., */precepts* or */courses/archive/fall19/cos316*

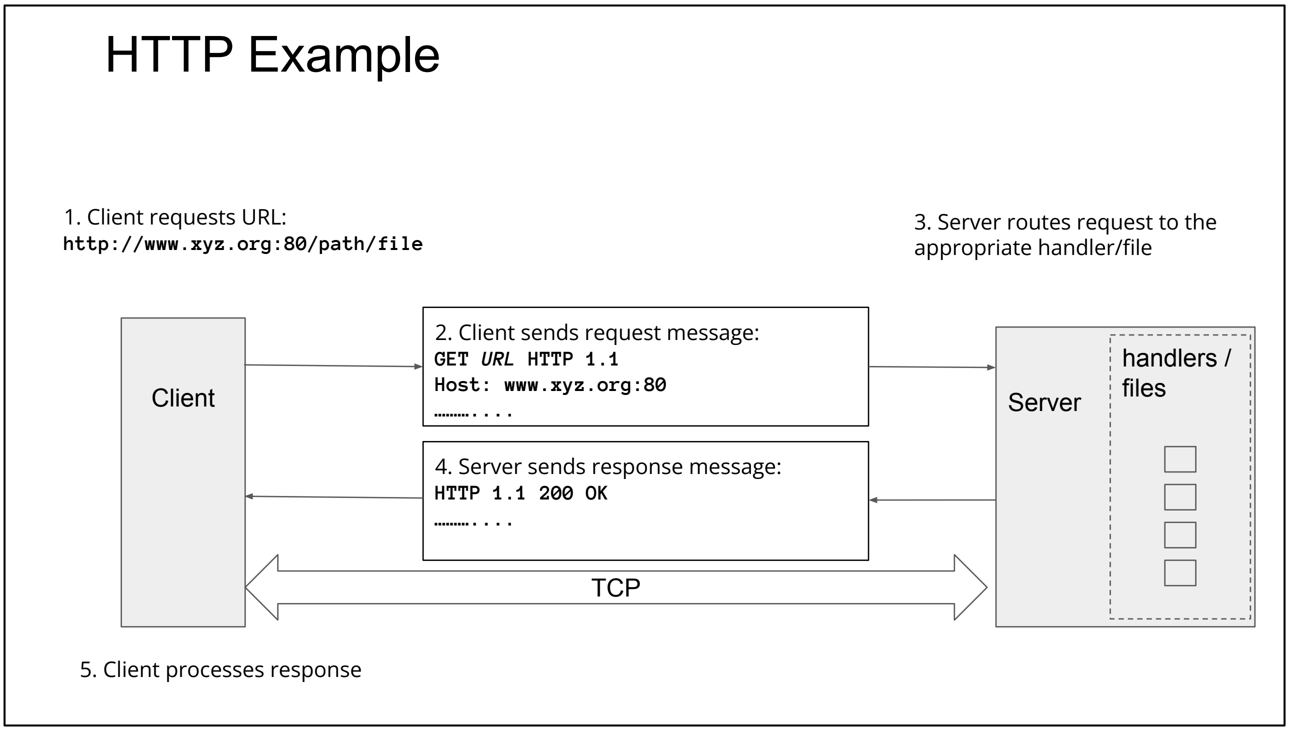
Parameters:
Encode additional information sent to the server. Behavior depends on the server.
E.g., *?mobile=true&lang=es*

Anchor:
Encode additional information for the client (not sent to server).
E.g., *#section-assignments*

Examples:

```
http://www.ietf.org/rfc/rfc959.txt
http://xyz.org:8081/route/subroute
http://www.ietf.org/rfc/rfc959.txt

mailto:ak18@cs.princeton.edu
ftp://tug.ctan.org/pub
rtsp://192.168.0.164/axis-media/media.amp
```

- URLs refer to uniform resource locator
  - “Uniform” means that it can interact with different resource such as file/webpage/email address
- Scheme:
  - Specify the protocol used when client is interacting with the server
  - HTTP: Use HTTP protocol to communicate. Usually webpage
  - FTP: File transfer protocol. Transfer files between clients and server
- Authority:
  - Location of the resource in terms of a service
  - Can be either an ip address or hostname. Port number is optional
  - If doesn’t provide port number, a default value will be used. Eg. HTTP -> 80. HTTPS -> 443
- Path:
  - The path specifies the location of the resource on the server.
- Parameters
  - It’s server dependent. Typical after a “?” mark and they are key=value pairs
  - lang=es means language is English. mobile=True means client is a mobile device.
- Anchor
  - This is to encode additional information. This will not send to the server
  - E.g., #section-assignment mean that after your browser received the page (web page), it points you to a specific part in this web page (Maybe that part is called section-assignment)



HTTP Example

1. Client requests URL:
`http://www.xyz.org:80/path/file`

2. Client sends request message:

```
GET URL HTTP 1.1
Host: www.xyz.org:80
………....
```

3. Server routes request to the appropriate handler/file

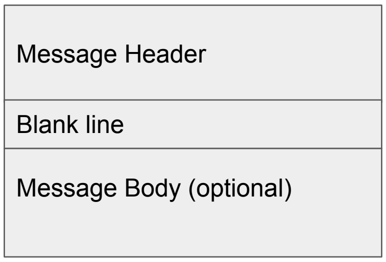
4. Server sends response message:

```
HTTP 1.1 200 OK
………....
```

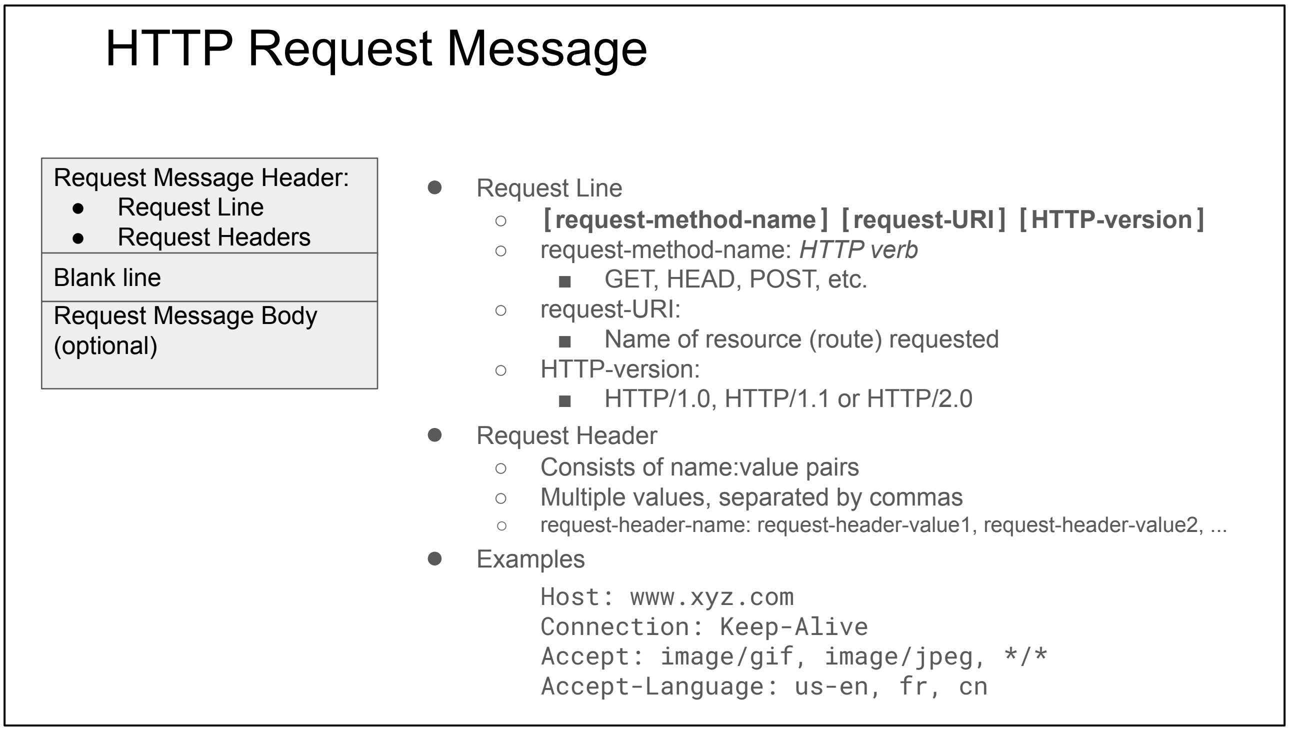
5. Client processes response

- Client requests URL
    - Specify the URL (resource) the user wants from the server
- Send request
    - Send HTTP request to the server and wait for the server's response
    - In this example, the request is a "GET", requesting the resource from the server
- Server routes request
    - The server basically looks for the right resource that the client requests
- Server sends response
    - After server finds the corresponding resource, it gives response.
    - 200 OK means that it finds the request is successful. The requested resource is also sent back in this response.
- Client processes response
    - One request-response pair is finished.

HTTP Request and Response Messages



- This is a high-level overview of the structure of an HTTP request or response



HTTP Request Message

Request Message Header: • Request Line • Request Headers
Blank line
Request Message Body (optional)

- Request Line
  - **[request-method-name] [request-URI] [HTTP-version]**
  - request-method-name: *HTTP verb*
    - GET, HEAD, POST, etc.
  - request-URI:
    - Name of resource (route) requested
  - HTTP-version:
    - HTTP/1.0, HTTP/1.1 or HTTP/2.0
- Request Header
  - Consists of name:value pairs
  - Multiple values, separated by commas
  - request-header-name: request-header-value1, request-header-value2, ...
- Examples

```
Host: www.xyz.com
Connection: Keep-Alive
Accept: image/gif, image/jpeg, */*
Accept-Language: us-en, fr, cn
```

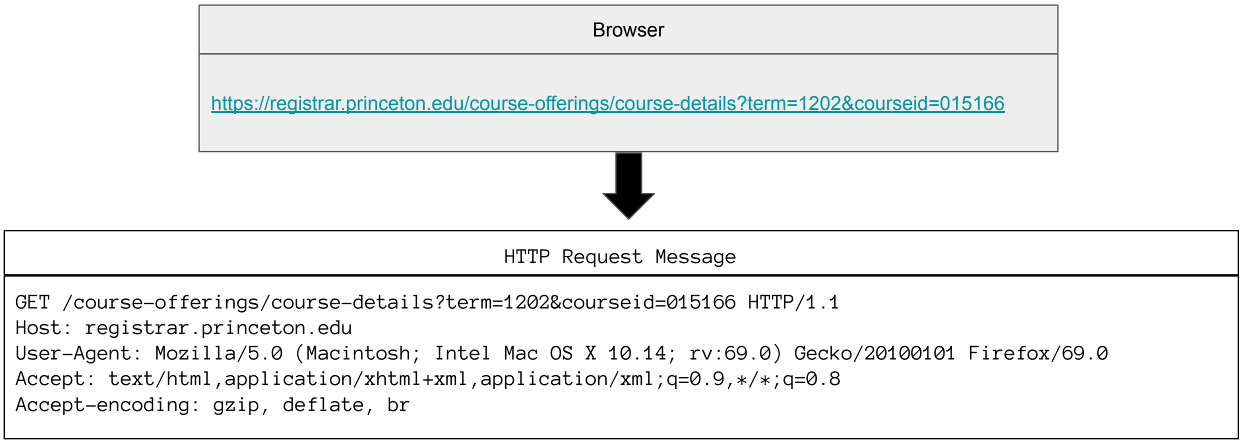
Request Line:
[request method name]:
1. GET: request resource from the server
2. HEAD; request the metadata of the resource from the server
3. POST: send data to the server

Request header example
- Host: www.xyz.com
  - Specify the hostname that the client will send the request to
- Connection: Keep-Alive
  - After this request-response pair, keep the connection open so that the TCP connection can be continuously used for the subsequent requests
- Accept: image/gif, image/jpeg, */*
  - Tell the server what type of media that the client can received. */* is a wildcard for any type
- Accept-Language: us-en, fr, cn
  - Preferred language. (US English, Franch, Chinese in this example)

HTTP Request Methods (*verbs*)

- Common methods
  - GET
    - retrieve a resource from the server
  - HEAD
    - return only the headers of GET response
  - POST
    - create a resource on the server (client sends resource in the request body)
- Case Sensitive

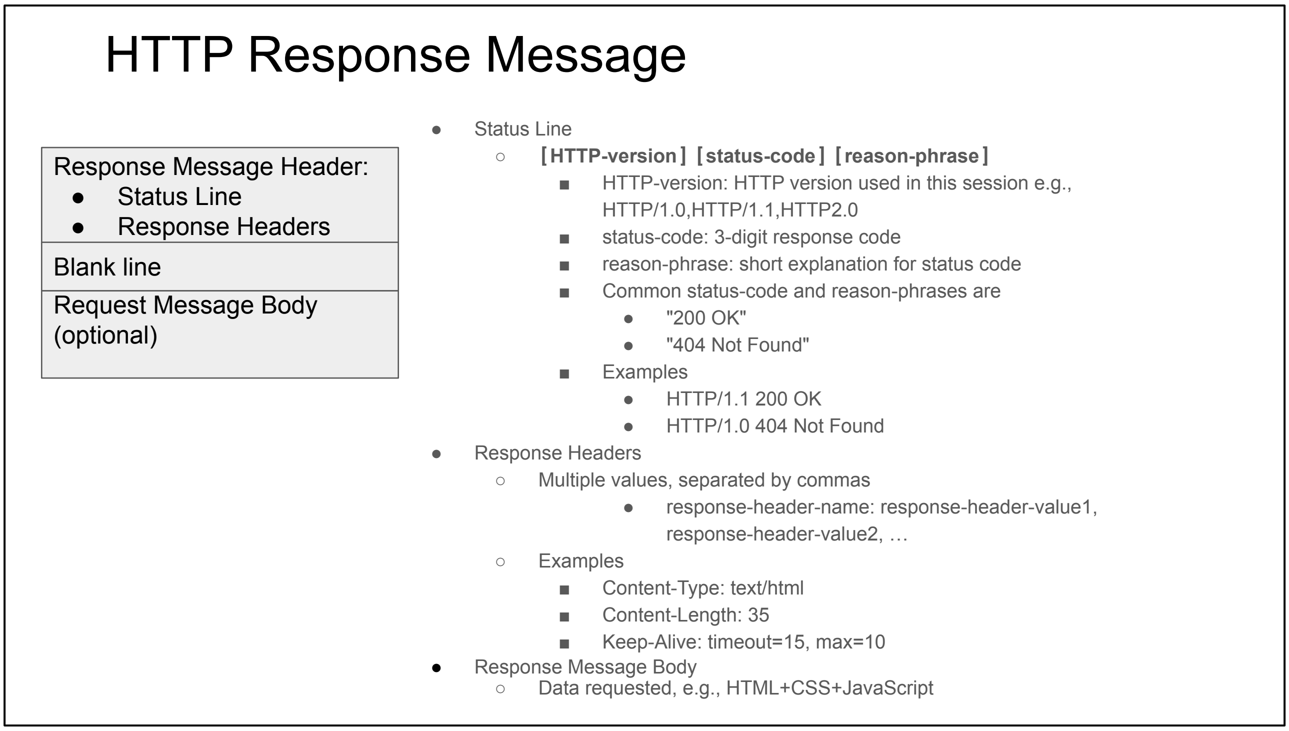
HTTP Request Message


Browser
https://registrar.princeton.edu/course-offerings/course-details?term=1202&courseid=015166
HTTP Request Message

```
GET /course-offerings/course-details?term=1202&courseid=015166 HTTP/1.1
Host: registrar.princeton.edu
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:69.0) Gecko/20100101 Firefox/69.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-encoding: gzip, deflate, br
```

When you are typing the URL and “enter” in the browser, the HTTP request message is constructed.

- User-Agent:
  - Provides some info for the client such as browser, OS etc
- Accept:
  - Type of content the client can receive
- Accept-encoding:
  - Type of compression client can accept.
  - Compression can make the data transmission faster.



HTTP Response Message

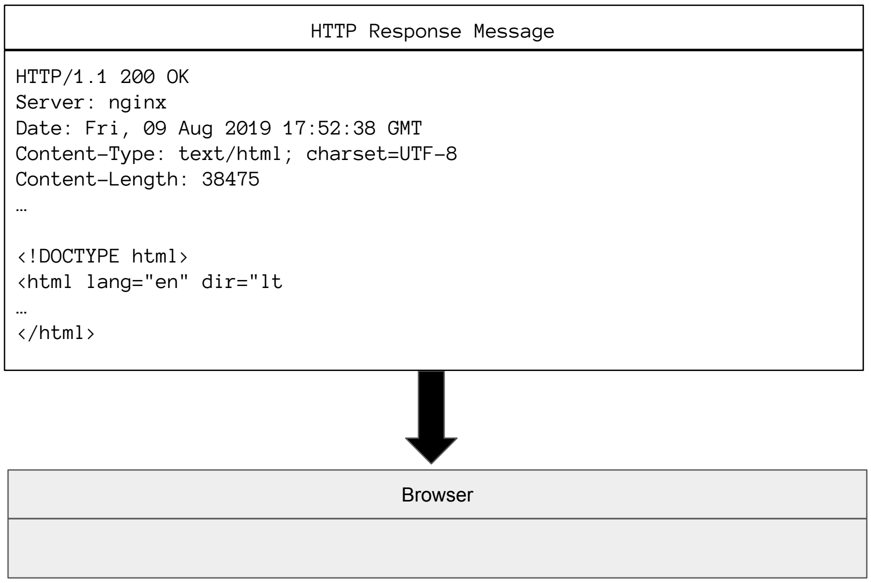
Response Message Header: ● Status Line ● Response Headers
Blank line
Request Message Body (optional)

- Status Line
  - [HTTP-version] [status-code] [reason-phrase]
    - HTTP-version: HTTP version used in this session e.g., HTTP/1.0,HTTP/1.1,HTTP2.0
    - status-code: 3-digit response code
    - reason-phrase: short explanation for status code
    - Common status-code and reason-phrases are
      - "200 OK"
      - "404 Not Found"
    - Examples
      - HTTP/1.1 200 OK
      - HTTP/1.0 404 Not Found
- Response Headers
  - Multiple values, separated by commas
      - response-header-name: response-header-value1, response-header-value2, …
  - Examples
    - Content-Type: text/html
    - Content-Length: 35
    - Keep-Alive: timeout=15, max=10
- Response Message Body
  - Data requested, e.g., HTML+CSS+JavaScript

Response header:
- Content-Type
  - Response content type.
  - HTML in this example
- Content-Length
  - Size of the response body (bytes)
- Keep-Alive
  - Timeout=15: connection will keep opening for 15 seconds for the (request-response) pair to be finished
  - max=10: Maximum number of requests can be sent using this connection before it is closed.

Response message Body: Exact data requested by the client.

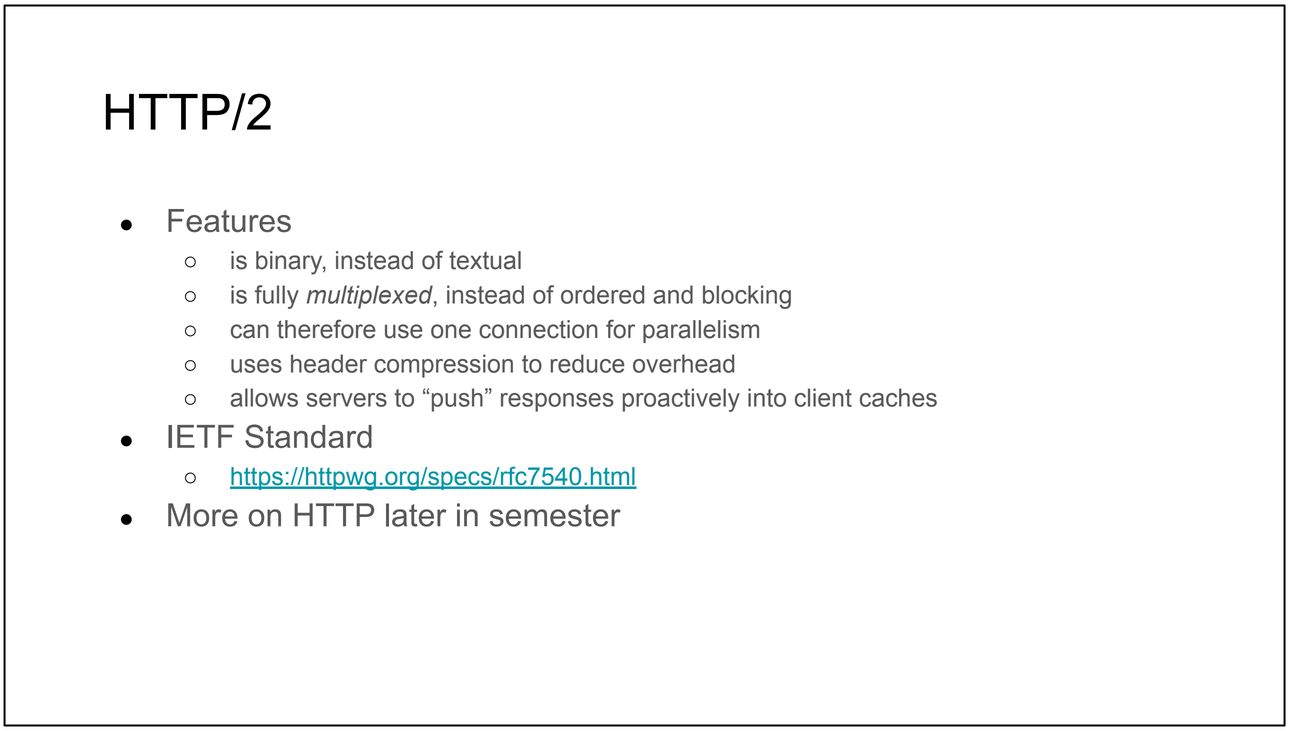
HTTP Response Message



```
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 09 Aug 2019 17:52:38 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 38475
…

<!DOCTYPE html>
<html lang="en" dir="lt
…
</html>
```

The client browser just received the HTTP response, parse it and render a webpage to the user.

The response body in this example is a html5 document.



HTTP/2

- Features
  - is binary, instead of textual
  - is fully *multiplexed*, instead of ordered and blocking
  - can therefore use one connection for parallelism
  - uses header compression to reduce overhead
  - allows servers to “push” responses proactively into client caches
- IETF Standard
  - https://httpwg.org/specs/rfc7540.html
- More on HTTP later in semester

HTTP/1.1 is created in 1997
HTTP/2 is created in 2015

HTTP/2 is much faster and more efficient. Significant performance improvement.