HTTP

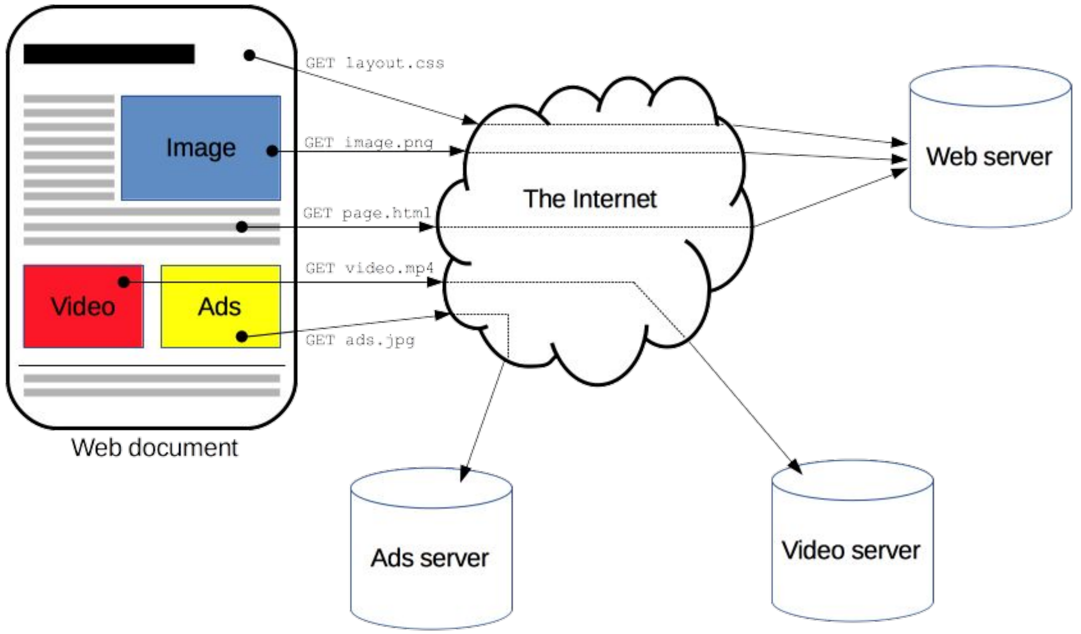
COEN 161

Intro to HTTP

- The ***HyperText Transfer Protocol*** was designed to retrieve web resources
- HTTP is a client-server protocol
- The messages sent by the client are called *requests*
- The messages sent by the server are called *responses*
- The client, aka the browser, initiates all requests

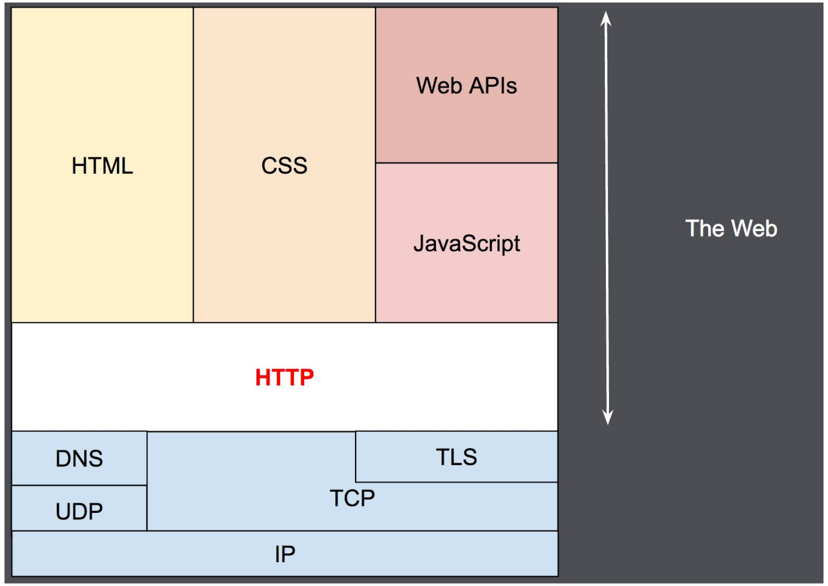
Intro to HTTP

- A complete document is made up of all the resources on the page



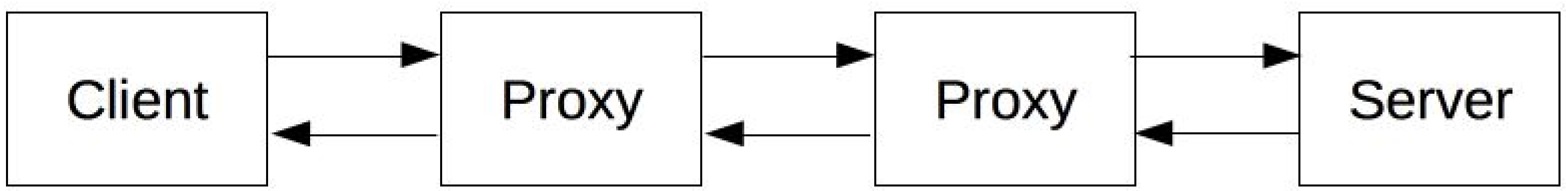
Intro to HTTP

- HTTP is an application layer protocol sent over TCP, a transport protocol
- HTTP uses a TLS-encrypted, reliable TCP connection



HTTP Components

- In an HTTP system, there is always a *client* and a *server*
- We may also see components in between called *proxies*



HTTP Client

- The client is also referred to as the *user-agent*
    - This is almost always the browser
    - It can also be an programs used by developers to debug applications
    - It can even a robot that crawls the web to maintain a search engine
- The browser **always** initiates request, it is **never** the server
- To present a webpage…
    - The browser sends a request to get the HTML document
    - It then parses the document and sends requests for any additional resources in the page
    - These may be CSS files, JS scripts, images, videos, etc.
- Anytime a hyperlink is clicked, the browser makes a request for the new page that was linked

HTTP Server

- A server can be a single machine, or a collection of machines that are under the same *host,* or domain
- This means that they are all under the same IP address
- When the client makes a request to the host, the server reads the request and responds accordingly

HTTP Proxies

- Due to the layered architecture of the web, a client rarely talks directly to a server
- There may be multiple things in between like routers, modems, and more
- Proxies perform specific functions, such as
  - Caching, saving content that is frequently requested
  - Filtering, like anti-virus scans, parental control, etc
  - Load balancing, to make the most efficient use of the network
  - Authentication
  - Logging, storing historical data

HTTP Basics

- HTTP is simple
  - HTTP messages are meant to be human readable
  - They only get encrypted when they are sent across the network
- HTTP is extensible
  - New functionality can be added through the use of *headers*
- HTTP is stateless, but not sessionless
  - Each request has no link to other requests
  - Order is not guaranteed when sending requests
  - However, *cookies* can be used to check for state
- HTTP requires a *reliable* connection
  - TCP is used since it is a reliable transport protocol
  - However, due to performance drawbacks, Google is researching and developing a new transport protocol, QUIC, based on UDP which is faster than TCP

HTTP Flow

- When the client wants to communicate with the server….
  1. The browser opens a connection using TCP or another reliable transport protocol
     - If there is an open connection available, it will reuse it
  2. The browser sends an HTTP request

```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

HTTP Flow

3. The browser reads the response from the server

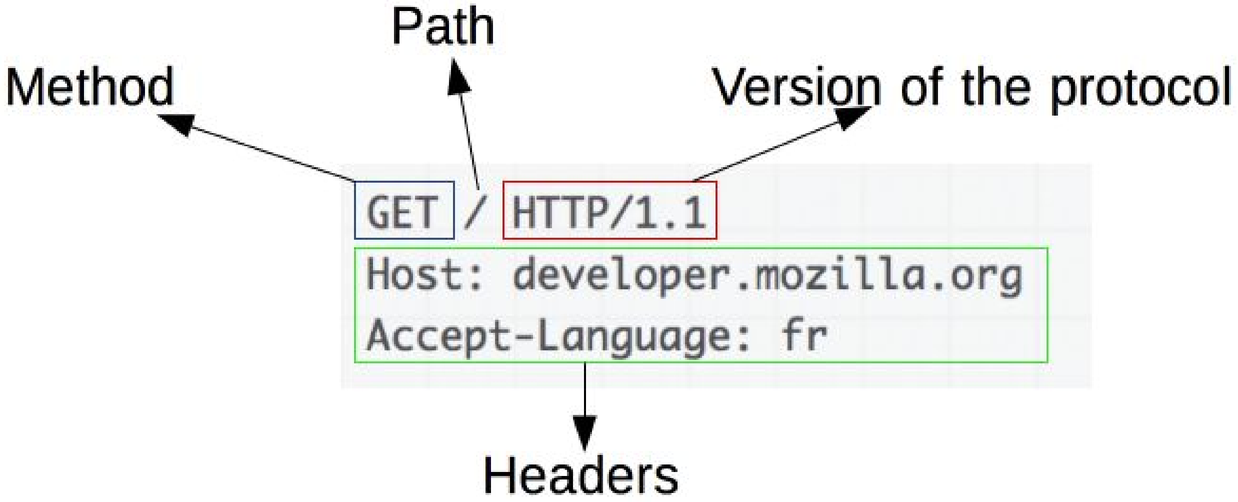
```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html… (all 29769 bytes of the requested web page)
```

HTTP Flow

4. Finally, the browser closes the connection
    - It may also save the open connection for reusing in later requests

HTTP Messages

- Before HTTP/2, all HTTP messages are human readable
    - Request



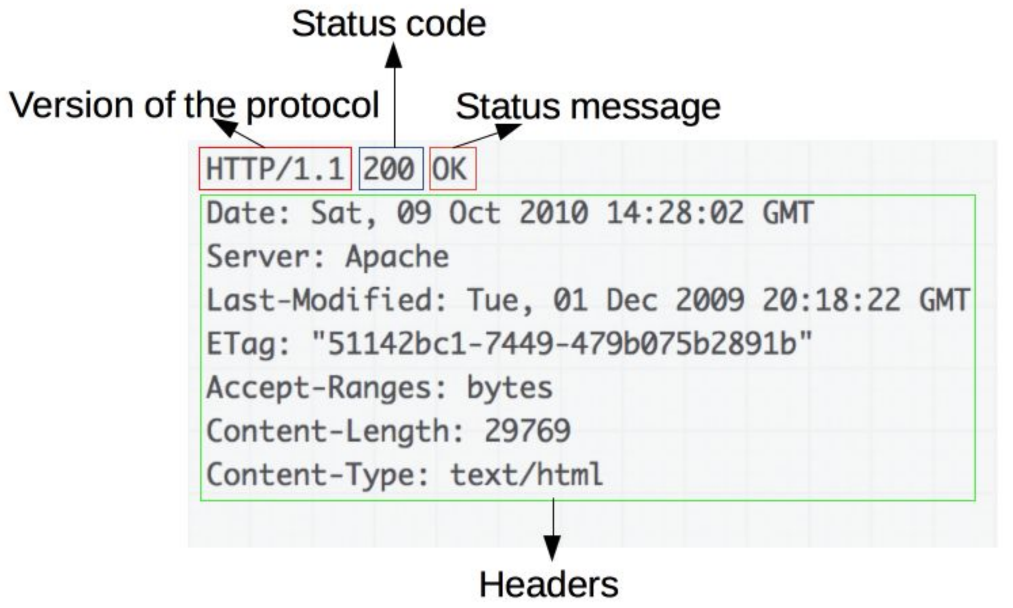
```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

HTTP Messages

- Request
    - Method
        - The method is always a *verb (GET, POST)* or a *noun (OPTIONS, HEAD)*
        - This defines the operation the client wants to perform
        - GET means the client is fetching a resource
        - POST means the client is posting a value from a form, for example
    - Path
        - The URL of the resource being requested
        - The elements may be stripped since they may be obvious based on the context such as the protocol since it should be HTTP, the domain and the port the port (the default is :80)
    - Version
        - The version of HTTP, here its HTTP/1.1
    - Headers
        - Additional parameters that the server may be expecting

HTTP Messages

- Response



```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
```

HTTP Messages

- Response
  - Version - same as the request
  - Status code
    - This indicates whether the request was successful (200) or not, and why
    - 400 codes indicate an error by the client, such as a bad path (404 - not found)
    - 500 codes indicate an error by the server
  - Status message - based on the code
    - 200 **OK**
    - 404 **Not Found**
  - Headers - same as requests

HTTP Messages

- Body
  - Some requests, such as POST, that are sending information to the server will have a message body
  - These are often just plain text, though it may be in a more meaningful format like JSON/XML
  - A response that returns information, such as GET, will also have a body
  - When retrieving a resource, like an HTML document, the body will be plain HTML, though it can also contain meaningful formats, such as JSON/XML, or even plain text

XMLHttpRequest

- XMLHTTPRequest is a JavaScript object designed to interface with HTTP on the client side
- It allows you to send and receive requests in JavaScript
- You must use it like a constructor

```
var xhttp = new XMLHttpRequest();

xhttp.open("GET", "ajax_info.txt", true);

xhttp.send();
```

XMLHttpRequest

- Receiving a response uses the same object

```
function loadDoc() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("demo").innerHTML =
            this.responseText;
       }
    };
    xhttp.open("GET", "ajax_info.txt", true);
    xhttp.send();
}
```

XMLHttpRequest

- readyState holds the current status of the request
    - 0: request not initialized
    - 1: server connection established
    - 2: request received
    - 3: processing request
    - 4: request finished and response is ready
- onreadystatechange is an event triggered when the readyState changes
- status holds the HTTP status code (200, 404, etc.)
- statusText holds the status message (OK, Not Found, etc.)
- responseText holds the body of the message

Resources

https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

https://www.w3schools.com/js/js_ajax_http_send.asp