# Web Programming

HTTP

# Introduction

- HTTP is the transfer protocol for web applications
  - H T T P: **H**yper **T**ext **T**ransfer **P**rotocol
  - HTTP 1.0 (RFC 1945), HTTP 1.1 (RFC 2068), HTTP 2 (RFC 7540)
  - In fact, it can be used to transfer everything (not only hyper text)
    - Text Documents e.g. HTML, XML, JSON, etc.
    - Multimedia e.g. JGP, GIF, MP4, MKV, etc.
    - Application e.g. PDF, ZIP, etc
- The foundation of any data exchange on the Web

# Introduction (Cont.)

- HTTP uses the client/server paradigm
  - HTTP server provides resource
  - HTTP client (usually web browser) gets resource
- But not pure client/server communication
  - Proxies
  - Caches
  - ...

# Introduction (Cont.)

- HTTP is an **application layer protocol**
- HTTP assumes reliable communication
  - over TCP
  - default (server) port: 80
  - client port is chosen randomly per connection
- HTTP is **Stateless**
  - Server does not keep history/state of client
  - High performance & Low Complexity
  - Problematic in some applications (sessions)
    - Cookies
    - JSON Web Tokens

# Data Resources to Transfer

- HTTP is the protocol to transfer data between server and client (usually from server to client)
- Which data?
  - It can be anything
  - In web, usually, it is a resource/object on server
- Each resource must be identified/located uniquely
  - URI (Uniform Resource Identifier)
  - URL (Uniform Resource Locator)
- URIs **identify** and URLs **locate**; however, locators are also identifiers, so

**Every URL is also a URI, but there are URIs which are not URLs.**

# URI in Action

- A Uniform Resource Name (URN) is a URI that identifies a resource by name in a particular namespace.
- International Standard Book Number (ISBN) system

# URL

<**protocol**(**scheme**)> :// <Domain Name> : < port> / <**path**> ?<**query**> #<**frag**>

- https://sbu.ac.ir/SitePages/Home.aspx
- https://www.bing.com/search?q=Hello+World&form=QBLH&sp=-1&pq=&sc=0-0&qs=n&sk=&cvid=7F2B496642F94D5F95989756B4FF60EE
- ftp://aeneas.mit.edu/
- https://en.wikipedia.org/wiki/Internet#Terminology

# URL (Cont.)

- **Scheme**: The application layer protocol
  - HTTP: The web protocol
  - HTTPS: Secure HTTP
  - FTP: File Transfer Protocol
  - File: Access to a local file
  - javascript: Run javascript code
  - mailto: Send mail to given address
  - etc.

# URL (Cont.)

- **Path**: The path of the object on host filesystem
  - E.g. web server root directory is /var/www/
    - http://www.example.com/1.html → /var/www/1.html
    - http://www.example.com/1/2/3.jpg → /var/www/1/2/3.jpg

# URL (Cont.)

- **Query**: A mechanism to pass information from client to active pages or forms
  - Fill information in a university registration form
  - Ask Bing! to search a phrase
- Starts with "?"
- name=value format
- "&" is the border between multiple parameters

# URL (Cont.)

- **Frag**: A name for a part of resource
  - A section in a document
- Handled by browser
  - Browser gets whole resource (doc) from server
  - In display time, it jumps to the specific part
- Try these out, by
  - https://en.wikipedia.org/wiki/RGBA_color_model#:~:text=While%20it%20is-,sometimes,-described%20as%20a (chrome-only)

# URL (Cont.)

- **Domain names** are case insensitive according to RFC 4343.
- The rest of URL is sent to the server via the GET method and etc. This may be case sensitive or not.

# URL (Cont.)

- URL is encoded by client before transmission
- How: Each byte is divided into two 4-bit group, hexadecimal of the 4-bits are prefixed by %
  - ~ → 126 → %7E
- What & Why?
  - Non-ASCII (e.g., Persian Characters, Emoji)
  - Reserved character when are not used for special role
  - Unsafe character, e.g. space, %, ...

https://www.example.com/search?q=coffee & tea

→ https://www.example.com/search?q=coffee%20%26%20tea

# URL in Action

- User asks the browser to retrieve a resource
- Browser finds the ip address of <host> (DNS lookup)
- Browser creates a TCP connection to the IP address and the <port>
- Browser sends http requests through the connection
- Browser gets the response and processes it

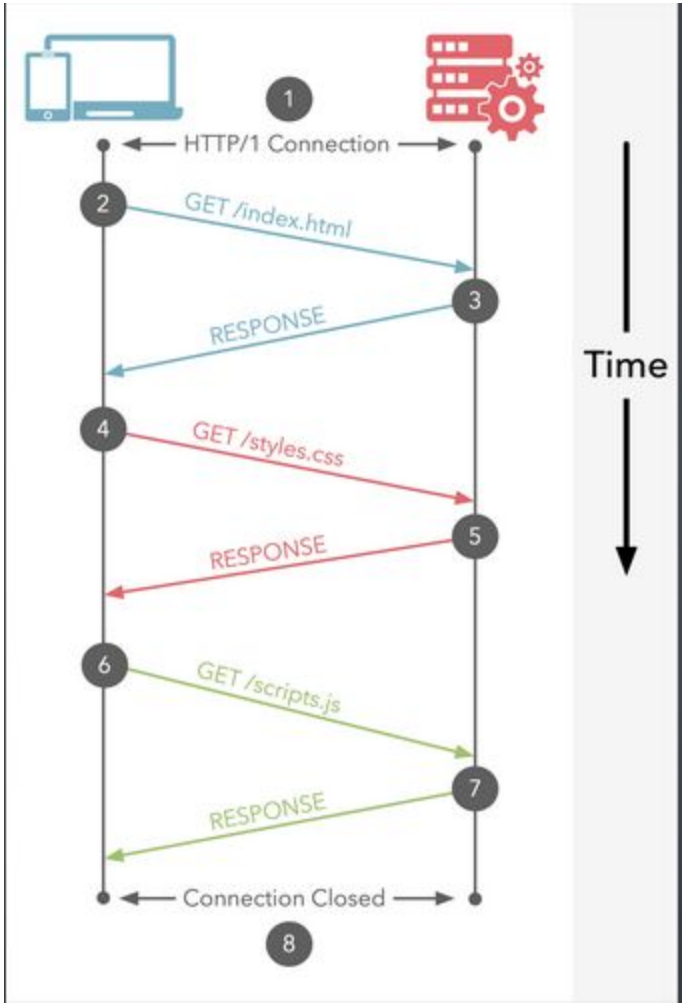
# URL in Action (Cont.)

sudo tcpdump --interface connectify0 --number -n -v 'port 80 and dst host 185.211.88.129'

# How does HTTP work? Transactions

- HTTP data transfer is a collection of transactions
- Each transaction is composed of 2 types of HTTP messages
    i. Requests are identified by methods
        - **Method**: The action that client asks from server
    ii. Responses are identified by status codes
        - **Status**: The result of the requested action

# HTTP Transactions



# HTTP Transactions in Web

- (Typically) each web page contains multiple resources
  - The main skeleton HTML page
  - Some linked materials: figures, videos, JS, CSS, etc.
- Displaying a web page by a browser
  - Get the HTML page (first transaction)
  - Try to display the page (rendering)
  - Other resources are linked to the page
  - Get the resources (subsequent transactions)

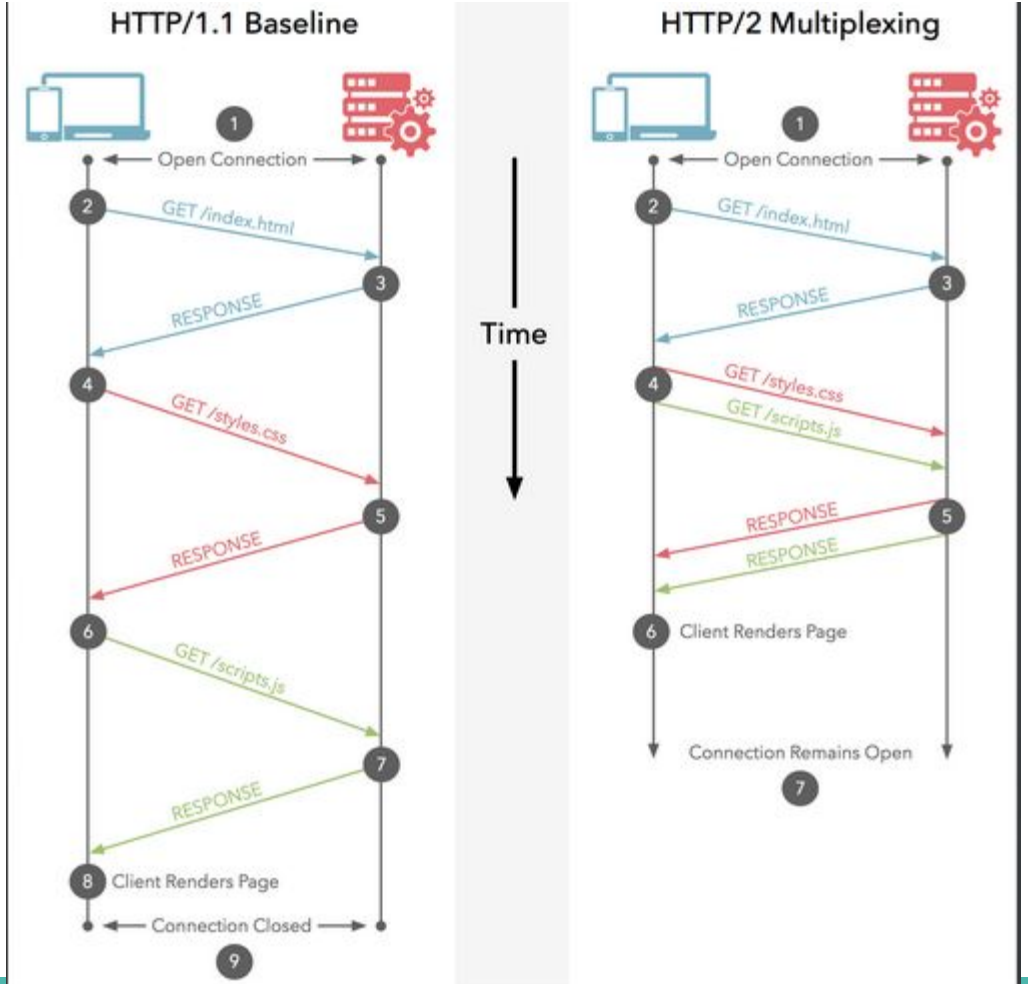
# HTTP Transactions in Web

- HTTP Transactions & TCP Connections (HTTP/1.x)
  - **Non-persistent (short-lived connection)**
    - A new TCP connection per HTTP request (object)
    - Network overhead + Connection establish delay + Resource intensive
    - Parallel connections speed up browsing
  - **Persistent**
    - Get multiple HTTP requests (objects) using a single TCP connection
    - No extra processing & networking overhead
    - Poor performance if implemented in serial manner
    - Pipeline requests speed up browsing (HTTP/1.1)

# HTTP/2 for a Faster Web

- HTTP Transactions & TCP Connections (HTTP/1.x)
  - HTTP Pipelining
    - Send successive requests, over the same persistent connection, without waiting for the answer.
    - it still allowed a single large or slow response to block all others that followed.
    - is complex to implement correctly:
- HTTP/2 Multiplexing
  - Multiplexing allows multiple request-response messages to be in flight over a single HTTP/2 connection, at the same time.

# HTTP/2 for a Faster Web

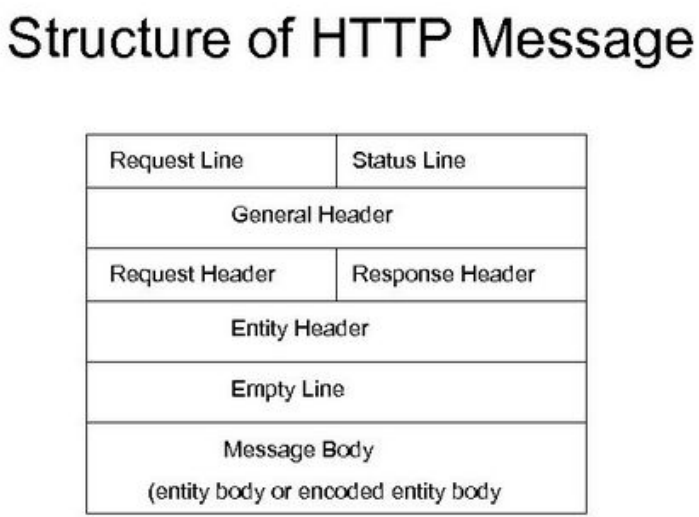


# HTTP Messages

- HTTP requests, and responses, share similar structure and are composed of 4 parts:
  - A **start-line** describing the **requests** to be implemented, or its **status** of whether successful or a failure. This start-line is always a single line.
  - An optional set of **HTTP headers** specifying the request, or describing the body included in the message.
  - A **blank line** indicating all meta-information for the request has been sent.
  - An **optional body** containing data associated with the request (like content of an HTML form), or the document associated with a response. The presence of the body and its size is specified by the start-line and HTTP header

# HTTP Messages (Cont.)

- HTTP is text-based protocol
  - Human readable headers
  - The header is composed of some lines



# HTTP Messages (Cont.)

- E.g. HTTP request message
- E.g. HTTP response message

```
GET /index.html HTTP/1.1
Host: www.aut.ac.ir
User-Agent: Mozilla/36.0
Accept-Language: en-us
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Date: Sun, 02 Oct 2018 20:30:40
Server: Apache/2.2.2
Last-Modified: Mon, 03 May 2017 10:20:22
Connection: keep-alive
Content-Length: 3000

data data data ...
```

# HTTP Request Message

- Start line (Status line)
  - HTTP Method
  - The Path of the protocol, port,
  - The HTTP version,
- Header
- Body

# HTTP Methods

- Methods are actions that client asks from server to do on the specified resource (given by the path parameter)
- Which actions?
  - Basic data communication operations
    - **Safe** operations
      - Get a resource from server
      - Send data to server
    - **Unsafe** operations
      - Delete a resource on server
      - Create/Replace a resource on server

# HTTP Methods (Cont.)

- Debugging and troubleshooting
  - Get information about a resource
  - Check what the server has got from a client
  - Get the list of operations which can be applied on a resource

# HTTP Methods (Cont.)

- **GET**: Retrieve resource from server
- **HEAD**: Similar to GET but the resource itself is not retrieved, just the HTTP response header
  - Useful for debugging or some other applications
- **POST**: Submit data to be processed by the specified resource
  - Data itself is enveloped in message body

# HTTP Methods (Cont.)

- **DELETE**: Remove the resource
  - Not popular in web, can be used in other applications
- **PUT**: Add message body as the specified resource to server
- **PATCH**: A PATCH request is considered a set of instructions on how to modify a resource. Contrast this with PUT; which is a complete representation of a resource.
- **TRACE**: Server echoes back the received message
  - For troubleshooting & debugging
- **OPTIONS**: Request the list of supported methods by server on the resource

# HTTP Response Message

- The message for the result/response of the requested action
- Which responses?
  - Basic responses
    - Success
    - Failure
      - Bad client request
      - Server problem
      - ...
  - Others
    - E.g., Redirection to other resources

# HTTP Response Message

- Start line (Status line)
  - The protocol version, usually HTTP/1.1.
  - A status code, indicating success or failure of the request. Common status codes are 200, 404, or 302
  - A status text. A brief, purely informational, textual description of the status code to help a human understand the HTTP message.

E.g., HTTP/1.1 404 Not Found.

# HTTP Status Code

- 2xx
  - Successful responses
  - 200: OK
  - 201: Created
  - 204: No Content
- 4xx
  - Client errors
  - 400: Bad Request
  - 401: Unauthorized (Authorization required)
  - 403: Forbidden
  - 404: Not Found
  - 405: Method Not Allowed

# HTTP Status Code (Cont.)

- 5xx
  - Server errors
  - 500: Internal Server Error
  - 501: Not Implemented
  - 503: Service Unavailable
- 3xx
  - Redirects
  - 301: Moved Permanently
  - 302: Found
    - redirect status response code indicates that the resource requested has been temporarily moved to the URL given by the Location header
  - 307: Moved Temporarily
    - Resource has been moved, Redirection
    - Location header contains the new location of resource
  - 304: Not Modified

# HTTP Status Code (Cont.)

- 1xx
  - Informational responses
  - 101: Switching Protocol
    - This code is sent in response to an Upgrade request header from the client, and indicates the protocol the server is switching to.

# HTTP Headers

- Headers are additional information that is sent by client to server and vice versa
  - Most (almost all) are optional
- Which headers?
  - Information about client
  - Information about server
  - Information about the requested resource
  - Information about the response
  - Security/Authentication
  - ...

# HTTP Headers

- General headers
  - Appear both on request & response messages
- Request headers
  - Information about request
- Response headers
  - Information about response
- Entity headers
  - Information about body (size, ...)
- Extension headers
  - New headers (not standard)

# General Headers

- **Date**: Date & Time that message is created
- **Connection**: Close or Keep-Alive
  - Close: Non-persistent connection
  - Keep-Alive: Persistent connection
- **Via**: Information about the intermediate nodes between two sides
  - Proxy servers

# Request Headers

- **Host**: The name of the server (required, why?)
- **Referrer**: URL that contains requested URL
- Information about the **client**
  - User-Agent: The client program
  - Accept: The acceptable media types
  - Accept-Encoding: The acceptable encoding
  - Accept-Language: The acceptable language

# Request Headers

- **Range**: Specific range (in byte) of resource
- **Authorization**: Response to the authenticate
  - Will be discussed later
- **Cookie**: To return back the cookies
  - Will be discussed later
- **If-Modified-Since**: Request is processed if the objected is modified since the specified time.
  - Used in Web Caching
  - Will be discussed later

# Response Headers

- **Server**: Information about server
- **WWW-Authenticate:** Used to specify authentication parameters by server
- **Proxy-Authenticate:** Used to specify authentication parameters by proxy
- **Set-Cookie:** To send a cookie to client
- **Location:** The location of entity to redirect client
- **Last-Modified**: The date and time of last modification of entity
- **Content-Range**: Range of this entity in the entire resource
- **Expires:** The date and time at which the entity will expire

# Entity Header

- **Content-Length**: The length of body (in byte)
- **Content-Type**: The type of entity
  - MIME types: text/xml, image/gif
- **Allow**: The allowed request methods can be performed on the entity
  - This is in response of OPTIONS method

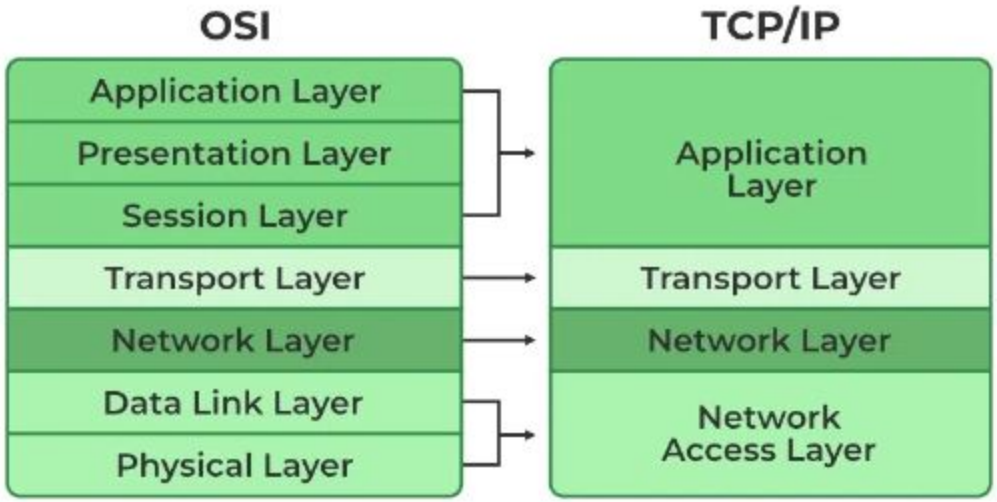
# Extension Headers

- Custom proprietary headers have historically been used with an X- prefix, but this convention was deprecated in June 2012
- implementation-specific and private-use parameters could at least incorporate the organization's name
  - ExampleInc-foo
  - VND.ExampleInc.foo (vnd stands for vendor)
- or primary domain name
  - com.example.foo
  - http://example.com/foo

# Resources

- The majority of these slides have been adapted from materials provided by other instructors:
  - lecture notes developed by Mr. Parham Alvani
- Other Resources
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/

# More Information




*TCP/IP and OSI*

Source: https://www.geeksforgeeks.org/tcp-ip-model/