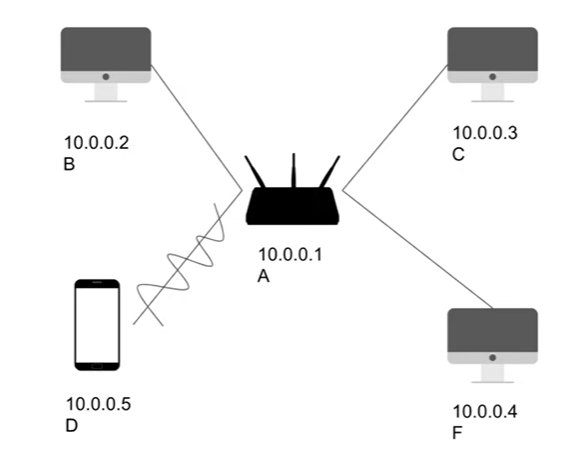
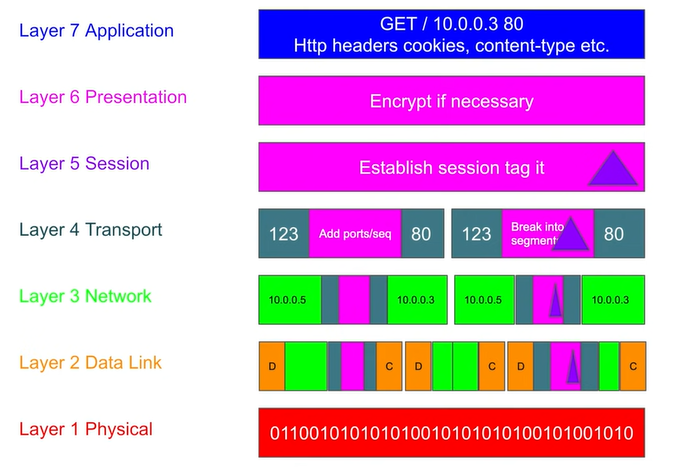
**OSI Model**

* Open Systems Interconnection Model
* 
* In the figure above, let’s assume that the computers are connected via ethernet and the mobile is connected to the router via wifi
* Each device has its IP address
* The letters represent the MAC (Media Access Control) Address, which is actually a unique 48-bit address that identified your network card (in other words, identifies your machine)
* Say, we have an application that runs on a server hosted on **C(Port 80)** and I wish to access an HTML page hosted on this server via the mobile phone (**D**)
* **Application Layer**
  + Mobile phone is the client, enters the url, sends a GET request to the server
  + As part of the request, a lot of content is sent, including headers, cookies, content-type, etc. It could have a body in the case of other request types. HTTP or another protocol that we are making use of prepares this string full of content and sends it over. This is the job of the **Application Layer(Layer 7)**.
  + So where is this string of data (which is converted into numbers which is converted into bits) actually sent?
* **Presentation Layer**
  + The data from Layer 7 is sent to **Presentation Layer (Layer 6)**. In this layer, the data that we receive from above is encrypted, if required (like in HTTPS or TLS, but in HTTP it does not need encryption)
* **Session Layer**
  + The next layer (**Session layer, Layer 5**) is a connection-oriented layer. Here, the data is tagged with a session ID to uniquely identify it. Session Layer cannot read the data as it is encrypted
* **Transport Layer**
  + Now, we have a bunch of bits, large in size. This layer breaks the data into smaller segments, and each segment is tagged with the source port number (suto-generated by the application) and the destination port number. Along with this, a sequence number is also tagged with the segment so that we can re-arrange the segments in the correct order
* **Network Layer** 
  + Segments are passed down into the network layer. This layer is not aware of the details of the ports and other things, it is just aware of the segments, to which it attaches 2 things:

1. Source IP address
2. Destination IP address

* This layer blindly sends these packets of data. Each packet is then passed down to Layer 2
* **Data Link**
  + This layer is related to the physical machine. It takes the packets and breaks them down into frames, then adds some headers with target MAC address to each of these packets, so that we know which machine to send it to.
  + Sometimes we do not know the MAC address, just the IP Address. That is where ARP (Address Resolution Protocol) comes in, which reverse engineers the IP address into the MAC Address
* **Physical Layer**
  + This layer on our device converts the bits of data (the frames) into electric/light/wifi/radio signals and sends them into the respective connection, but it does not know where to send them exactly, as electricity, light, etc does not have a specified direction. So, it goes everywhere. The data goes to all the servers, routers, and even itself.
* 
* The physical layer of the receiving device receives it and sends it to the data link layer.
* Data link layer checks the frames, and checks if it is supposed to receive them. If it is not supposed to, it will abort the process/drop the frames as the Network Card in the device discards them. If it is supposed to (this is determined by checking the headers for the destination address),
* Then the process moves forward, and goes to the upper layers, removes the headers in the network layer
* Then transport layer checks the ports and sees whether there is an application running on the destination port. It also keeps the information about the source port so that it can send a response/acknowledgment
* Session ID confirms the validity of the current session and lets the data go through if the session ID matches
* The data is then decrypted in the presentation layer (if necessary)
* Application layer will then analyse the request and do the necessary operations

**TCP/IP Model**

* Transmission Control Protocol / Internet Protocol
  + Two different protocols used together for making connections over the internet
  + TCP works on transport layer
  + IP works on network layer
* Proposed and developed by ARPA for military connections because our devices were platform-dependent, which means that a machine created by a company would only communicate with another device from the same company.
* OSI Model (by ISO) l is considered to be the inspiration for TCP/IP (by ARPA) but the two are independent of each other.
  + The idea behind the two models was to make communication platform-independent
  + OSI is theoretical, TCP/IP is focussed on practical implementation
* OSI Layers (7 layers)
  + Application
  + Presentation
  + Session layer
  + Transport
  + Network
  + Data Link
  + Physical layer
* TCP/IP layers (5 layers)
  + Application layer (Combining Application, Presentation, Session)
    - (Information stays in the form of **Data** in this layer)
  + Transport
    - (Data is converted into **Segments)**
  + Network
    - (Segments are converted into **Packets)**
  + Data link layer
    - (Packets are converted into **Frames**)
  + Physical layer
    - (Frames are converted into **Bits,** after which is travels)
* Another representation of TCP/IP layers
  + Application layer (Combining Application, Presentation, Session)
  + Host to host layer (Transport layer)
  + Internet layer (Network layer)
  + Network access layer (Combining Physical and Data link layer)
* TCP/IP features that supported decentralisation
  + End node verification: Instead of a central system ensuring successful data transfer, it was the two endpoints that were communicating, who were responsible for successful transfer of data.
    - This also means that information of the various routers via which our data is travelling is not relevant
  + Dynamic routing: End nodes have the capability to transfer data over multiple paths. The network chooses the path that is the best for the current operation
* Main network issues are:
  + Addressing: Resolving IP Address when we have MAC address and vice versa
  + Routing: Deciding the best path for data transfer
  + Name resolution: Resolving IP using DNS
  + Flow and error control: E.g. incoming data transfer rate could be 10 mbps whereas our capacity to receive data is only 2 mbps, we also need to handle such issues
  + Interoperability: Devices of different types being able to communicate with each other irrespective of location, protocol, etc.
* The 4 layers of TCP/IP solved the above 5 problems
  + Application layer
    - The user is connected to this layer
    - Protocols present in this layer: HTTP, FTP, TFTP, Telnet, SSH, SMTP, SNMP, NTP, DNS, DHCP, NFS
    - Basically, every protocol that you need to access the internet is present in the application layer
  + Host to host / transport layer
    - Responsible for transporting data and communicating between the layers
    - Protocols present in this layer: TCP, UDP
    - TCP
      * It is connection oriented. Data is transferred after the sender and receiver are connected. Data is sent, acknowledgement is received for each packet.
      * Performs sequencing and segmentation of the data
      * Deals with acknowledgement
      * Controls flow of data
      * If a packet fails to deliver, it re-transmits the data
      * No packet loss, so used in apps like email
      * Used for non-real time applications
      * Slow wrt UDP
      * Reliable
    - UDP
      * Not connection-oriented protocol like TCP
      * Used for real-time applications, e.g. video call, streaming
      * There can be packet loss
      * There is no re-transmission in case a packet is dropped
      * No acknowledgement
      * Faster
  + Internet layer
    - Responsible for logical transmission of data over the network
    - Protocols present in this layer: IP, ICMP, IGMP, ARP, RARP
    - IP
      * Responsible for delivering packets from the source host to the destination host by looking up the IP Address in the packet header
      * Determines the IP Address of the sender and receiver
      * Handles routing
      * Connection oriented
  + Network Access Layer
    - Looks out for MAC Addresses
    - Responsible for converting data into frames (data link layer props), then bits (physical layer props)

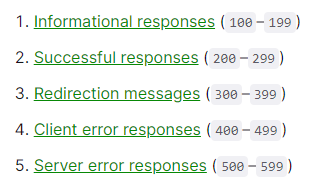
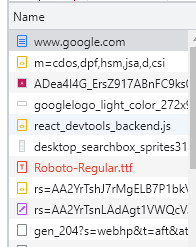
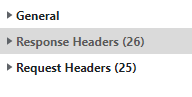
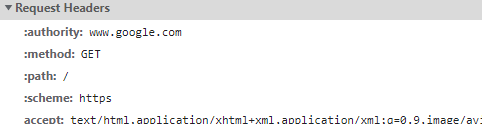
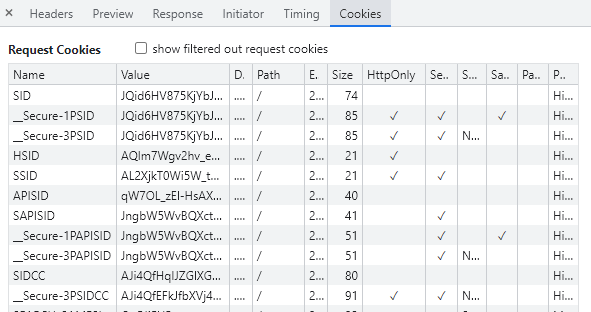
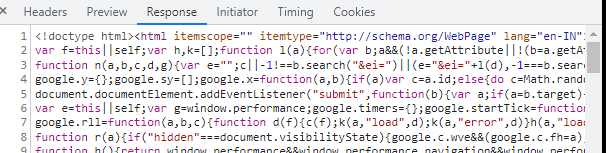
**The HTTP - HTTP/1.0, HTTP/1.1, HTTP/2**

* Hypertext transfer protocol
* The internet runs on it

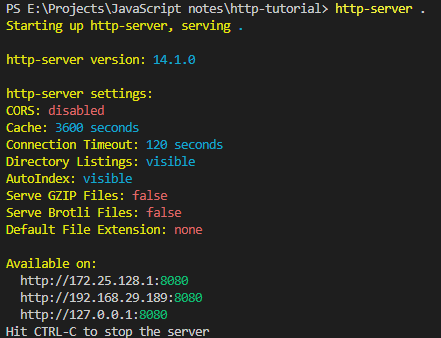
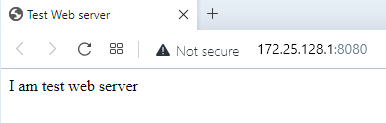
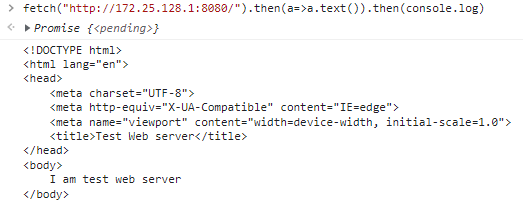
Client server architecture

* Client requests for something that is available on a certain server, the server sends the data as a response
* For HTTP, Browser is the client. It could also be Python or JS app
* Server is the HTTP Web Server (eg IIS, Apache Topcat, Node js, Python tornado)

HTTP Requests

* There are 4 major items in an HTTP request:
  + URL
    - It has a lot of components of its own
  + Method type
    - GET
      * For reading data
    - POST
      * For insertion of data
    - PUT
      * To replace the data with newly sent data
    - PATCH
      * For updating existing data
    - DELETE
      * For removal of data
  + [Headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers)
    - Includes host names, location, authorization tokens, etc
  + Body
    - Some data passed with the request
    - GET does not have a body, you can send data with a get request via a url
* HTTP Response
  + Has the following parts
    - [Status codes](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)
      * 
    - Headers
      * Application details (application/html, etc)
    - Body
      * The actual content that we requested, JSON, HTML file, etc.
  + **Demo**: Go to any website, open DevTools, open Network Tab
    - Refresh the page
    - 
    - A lot of rows will show up under the heading name
    - Click on any row, you will see data in 3 categories
    - 
    - In request headers
    - 
    - Notice the URL([www.google.com](http://www.google.com)) on which we have sent the request, the method (GET) as we are trying to open the website, and are therefore reading the data, and the path (‘/’).
    - There are many other details in this section
    - In response headers
    - 
    - This is because we are trying to get an html page
    - Try to find an image, and check its content type
    - In the general section, you will see similar details, which also include the status code
    - 
    - Try to analyse status codes on other content
    - You can even view the cookies
    - 
    - The response tab will have the html of the page that we have requests
    - 

Demo: Spin up your own server

* Install http-server 
* Make an html page, make some random changes to it
* Start the server in the following way
* ****
* Go to the URL
* 
* Try to make a request on this site in the browser itself using fetch api
* 

How does HTTP work?

An application (client) makes a request to the server.

* HTTP is a Layer 7 (Application layer) protocol of the OSI Model
* The internet is actually connected to Layer 1 (Physical Layer) where data is a bunch of 1s and 0s travelling as electricity/light
* The destination machine is responsible for parsing it, which starts from Layer 2, where switches uses metadata in the frames such as MAC Address to parse it, then pass it to Layer 3
* Layer 3 (Network layer) shows the IP Addresses, layer 4 (transport layer) shows us ports. The port used by HTTP is 80
* Finally, after moving through the rest of the layers, we move to layer 7, where we have the data unpacked by the previous layers, which is basically a blob string that looks like an HTTP request
* HTTP uses TCP to carry the chunks of data and maintain its integrity
* A 3 way handshake is established so that the TCP connection can be started
  + We open the connection between the client and the server (say by going to a website)
  + The request will be received by the server
  + The server will process the request, send the response with headers, data etc.
  + The connection is closed
* HTTPS is very similar to HTTP, except there is a TLS handshake happens in the beginning
  + This is to ensure that the the client and the server are authenticated, i.e. they have the same key
  + Client sends the encrypted request
  + Encrypted Response is sent back to the client
  + Connection is closed

HTTP 1.0

Back in 1996, maximum RAM was 64 MB was available and TCP connections took up a bit on memory and it was not advisable to leave the connections open

So the idea used for to:

1. Open connection
2. Send request (from client)
3. Send response (from server)
4. Close connection

There was not a lot of data in the html files, and mainly consisted of text.

We repeated the process for every html page, image, etc.

This proved to be expensive as there was a new TCP connection with every request, it was slow, and there was a lot of buffering.

This lasted for about a year

HTTP 1.1

This lasted for about 20 years.

In the headers, there was a “Keep-alive” header, which was an indication to the server to not close the connection.

So everytime a new request was made, a new connection was not needed to be established, we just used the old one. After we were done, the connection was closed.

More concepts such as caching and e tags were introduced as well

So the main points for this version of HTTP were:

1. Persistent TCP connection
2. Low latency
3. Streaming (transferring data in chunks)
4. Pipelining (disabled by default, what it did was send all request together instead of waiting for a response before sending the next request)

HTTP 2

Main ideas:

1. Compression
2. Multiplexing
   1. A lot of requests are shoved into one channel. Client will combine multiple TCP packets as one request. This solved problems caused by pipelining
3. Server Push
   1. Performance technique aimed at reducing latency, where the server sends some data to the client before the client asks for it
4. SPDY
   1. SPDY(speedy) is a network protocol which manipulates http protocol by compressing headers, predicting clients requests and other techniques, to fasten web experience. SPDY was developed by Google.
5. Secure by default
6. Protocol negotiation during TLS (NPN/ALPN)
   1. As there are a lot of old servers, so we have to negotiate the protocol with those servers to make HTTP 2 work on them
   2. This protocol is called Next Protocol Negotiation, replaced by Application Layer Protocol Negotiation
   3. The client mentions the list of protocols that it supports, the server picks the best option during the TLS handshake.
   4. ALPN is an extension of TLS, therefore the process happens quickly

HTTP 3 (Experimental)

TCP is slow, therefore HTTP 3 is meant to replace TCP with QUIC (UDP with congestion control)

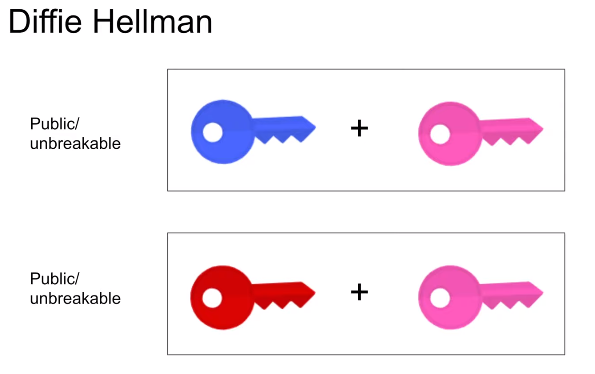
It is lossy

It has all the features of HTTP 2

**TLS/SSL - Cryptographic protocols for secure communication**

**between web servers & client**

**Transport Layer Security, TLS 1.2 and 1.3**

* Protocol for securing communication between client and server
* Used by HTTPS
* HTTP runs on TCP, which is a stateful protocol, with bidirectionality, i.e. it has knowledge of both client and server
* HTTP is a stateless protocol, and it just needs TCP to transfer data
  + We have studied that in HTTP:
    - TCP Connection is opened
    - Request is made
    - Response is sent by server with headers, cookies, required data, etc
    - Connection is eventually closed
* HTTP is very similar, the difference is:
  + Default port is 443 (as compared to 80 in HTTP)
  + Before a request, there is a handshake to make the client and server agree on the (symmetric) key which is to be used to encrypt and decrypt data
  + Once both client and server have a key, the client will take the key, and encrypt its request with it
  + TCP connection will take this encrypted request and transfer it to the server
  + Server will decrypt and process the request, prepare the response and encrypt it
  + Response is sent back to the client. Any hacker who tries to access the data will receive the encrypted response which will appear as garbage value
  + The client (who has the key) will be able to decrypt the response.
* SSL is old, obsolete, and insecure
* TLS 1.2
  + Focus on the handshake mentioned previously
    - When TLS was being designed, the idea was to make it configurable, giving client and server options
    - Client sends a “Client hello”
      * The client introduces itself to the server and gives info about the ciphers that it supports
    - The server gives the client its certificate (which is basically its public key), which can be considered “server hello”.
    - This certificate is used by the client to generate the pre master key which is used to generate the symmetric key
    - It will use the certificate to encrypt the key and send the key to the server (RSA model)
      * This is a problem because we are sending the key over the network
      * If someone managed to get the private key of the server, the operation is compromised
    - Server receives the key, will let the client know
    - Now, communication of data can begin
    - Now both the client and server have the symmetric key, they will send encrypted request and response
    - Because of the preliminary process that take place before data exchange, it makes the whole process slow
    - The key sending to the server is insecure as mentioned earlier
  + **Diffie Hellman**
    - Problem with key exchanging algos like RSA is that the op is compromised if server’s private key is leaked
    - To solve it, the idea was to send information about the key, instead of the key itself
    - We generate 3 keys: 2 private keys and 1 public key, and you do not send the private keys over the network. The 3 keys together make up the secret key
    - 
    - Each private key can be sent in combination with the public key, and it is very expensive to break the encryption
* **TLS 1.3**
  + Took away the customization that 1.2 offered
  + The client will be given the choice of symmetric algorithm but is required to always use using Diffie Hellman for key exchange
  + Client generates the public and the private key
    - Merges the public key and private key and sent it over
  + Server generates its own private key, takes the two merged keys, combines all of them, which gives us the symmetric key (actually it gives the input to the algorithm that gives us the symmetric key)
  + The client needs the symmetric key to decrypt data,for which it needs the server’s private key
  + Server merges the public key with its own private key and sends it
  + Now client has all 3 keys and can make the symmetric key
  + This process is done in fewer steps so it is faster
  + Now data exchange can take place