1. **Difference between HTTP1. 1 vs HTTP2**

|  |  |
| --- | --- |
|  | **HTTP1. 1** |
| **Multiplexing** | HTTP/1.1 loads resources one after the other,  so if one resource cannot be loaded, it blocks all the other resources behind it. |
| **Server push** | A server only serves content to a client device  if the client asks for it. However, this approach  is not always practical for modern webpages,  which often involve several dozen separate  resources that the client must request. |
| **Header compression** | Small files load more quickly than large ones.  To speed up web performance, both HTTP/1.1 and HTTP/2 compress HTTP  messages to make them smaller. |

|  |
| --- |
| **HTTP2** |
| HTTP/2 is able to use a single TCP connection to send multiple streams of data at once so that no one resource blocks any other resource. HTTP/2 does this by splitting data into binary-code messages and numbering these messages so that the  client knows which stream each binary message belongs to. |
| HTTP/2 solves this problem by allowing a server to "push"  content to a client before the client asks for it. The server also  sends a message letting the client know what pushed content to expect. |
| HTTP/2 uses a more advanced compression method called HPACK that  eliminates redundant information in HTTP header packets. This  eliminates a few bytes  from every HTTP packet. Given the volume of HTTP packets involved  in loading even a single webpage, those bytes add up quickly, resulting in faster loading. |

1. **http version history**

HTTP has four versions — HTTP/0.9, HTTP/1.0, HTTP/1.1, and HTTP/2.0. Today the version in common use is HTTP/1.1 and the future will be HTTP/2.0.

**HTTP/0.9 — The One-line Protocol**

1. Initial version of HTTP — a simple client-server, request-response, telenet-friendly protocol

2. Request nature: single-line (method + path for requested document)

3. Methods supported: GET only

4. Response type: hypertext only

5. Connection nature: terminated immediately after the response

6. No HTTP headers (cannot transfer other content type files), No status/error codes, No URLs, No versioning

Popular web servers (Apache, Nginx) still supports HTTP/0/9.

**HTTP/1.0 — Building extensibility**

1. Browser-friendly protocol

2. Provided header fields including rich metadata about both request and response (HTTP version number, status code, content type)

3. Response: not limited to hypertext (Content-Type header provided ability to transmit files other than plain HTML files — e.g. scripts, stylesheets, media)

4. Methods supported: GET , HEAD , POST

5. Connection nature: terminated immediately after the response

**Establishing a new connection for each request** — major problem in both HTTP/0.9 and HTTP/1.0

Both HTTP/0.9 and HTTP/1.0 required to open up a new connection for each request (and close it immediately after the response was sent). Each time a new connection establishes, a TCP three-way handshake should also occur. For better performance, it was crucial to reduce these round-trips between client and server. HTTP/1.1 solved this with persistent connections.

**HTTP/1.1 — The standardized protocol**

1. This is the HTTP version currently in common use.

2. Introduced critical performance optimizations and feature enhancements — persistent and pipelined connections, chunked transfers, compression/decompression, content negotiations, virtual hosting (a server with a single IP Address hosting multiple domains), faster response and great bandwidth savings by adding cache support.

3. Methods supported: GET , HEAD , POST , PUT , DELETE , TRACE , OPTIONS

4. Connection nature: long-lived

**Keep-Alive and Upgrade headers**

**Keep-Alive header**

1. The Keep-Alive header was used prior to HTTP/1.1 and was obsoleted by HTTP/1.1 making persistent connections the default behavior. Keep-Alive header can be used to define policies for long-lived communication between hosts (i.e. allows a connection to stay active until an event occurs). This laid foundation for persistence, reusable connections, pipelining, and many more enhanced capabilities in modern web communication protocols.

2. Client, server, or any intermediary can provide information for Keep-Alive header independently. Also, a host can add timeout and max parameters in order to set a timeout or limit maximum request count per connection.

3. HTTP pipelining, multiple connections, and many more improvements have been implemented, thanks to the Keep-Alive header’s behavior.

**Upgrade header**

1. With Upgrade header introduced in HTTP/1.1, it is possible to start a connection using a commonly-used protocol, such as HTTP/1.1, then request that the connection switch to an enhanced protocol type like HTTP/2.0 or WebSockets.

2. In an upgraded protocol connection, max parameter (maximum request count) is not present. The upgraded protocol can provide new policies for timeout parameter (if not specifically defined, it uses default timeout value in underlying protocol).

**HTTPS**

1. Hyper Text Transfer Protocol Secure (HTTPS) is the secure version of HTTP. It uses SSL/TLS for secure encrypted communications.

2. Originally developed by Netscape in mid-1990s, SSL (Secure Socket Layer) is a cryptographic protocol enhancement to HTTP, which defines how client and server should communicate with each other securely. TLS (Transport Layer Security) is the successor of SSL.

3. An HTTPS connection can protect the data transfer from the man-in-the-middle attacks and common security threats by providing bidirectional encryption for communications between a client and server.

**SSL/TLS Handshake** — major problem in HTTPS

1. Although HTTPS is secure by its design, the SSL/TLS handshake process consumes a significant time before establishing an HTTPS connection. It normally costs 1–2 seconds and drastically slows down the startup performance of a website.

**HTTP/2.0 and the future**

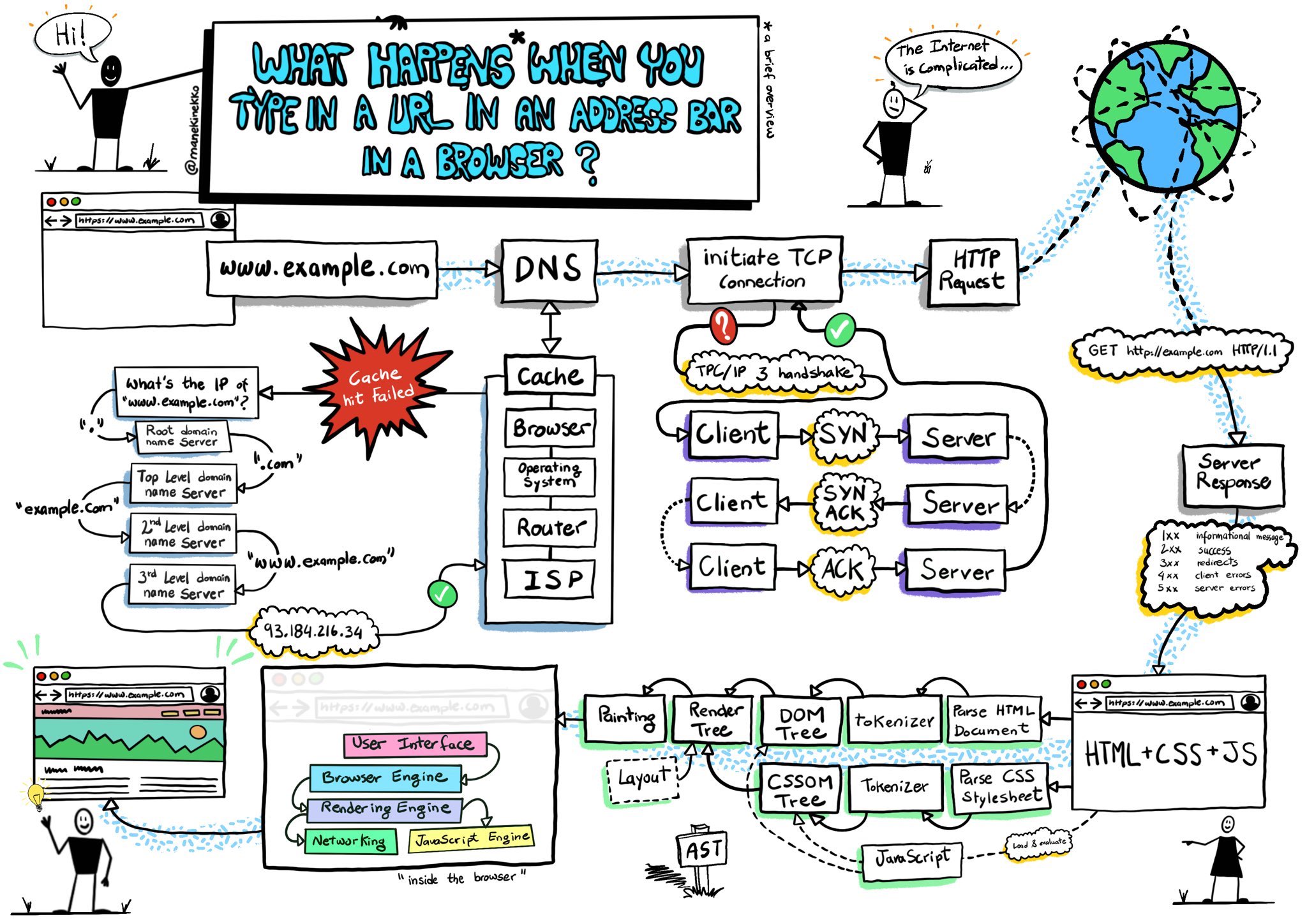
All above features are being used by major web servers and browsers today. But modern enhancements like HTTP/2.0, Server Side Events (SSE), and Websockets have changed the way that the traditional HTTP works. It has many advantages and performance benefits.

1. **List 5 difference between Browser js console vs Node js**

|  |
| --- |
| **Browser JS(console)** |
| In browser “window” is a predefined global object which has functions and attributes |
| In browser “location” is another predefined object |
| In browser “require” is not predefined object |
| In browser module is not required |
| In browser “document” is a predefined object |

|  |
| --- |
| **Nodejs** |
| Nodejs doesn’t have it |
| Nodejs doesn’t have it |
| Nodejs has it |
| Nodejs you have to keep your code inside the module. |
| Nodejs doesn’t have it. |

1. **what happens when you type a URL in the address bar in the browser**

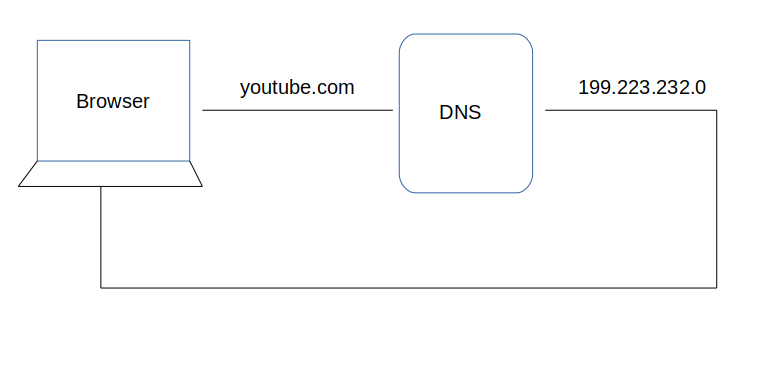


1. You enter a URL into a web browser
2. The browser looks up the IP address for the domain name via DNS
3. The browser sends a HTTP request to the server
4. The server sends back a HTTP response
5. The browser begins rendering the HTML
6. The browser sends requests for additional objects embedded in HTML (images, css, JavaScript) and repeats steps 3-5.
7. Once the page is loaded, the browser sends further async requests as needed.

Detail Explanation

**DNS lookup to find IP address**

After hitting the URL, the first thing that needs to happen is to resolve IP address associated with the domain name. DNS helps in resolving this. DNS is like a phone book and helps us to provide the IP address that is associated with the domain name just like our phone book gives a mobile number which is associated with the person’s name.



1. After hitting the URL, the browser cache is checked. As browser maintains its DNS records for some amount of time for the websites you have visited earlier. Hence, firstly, DNS query runs here to find the IP address associated with the domain name.

2. The second place where DNS query runs in OS cache followed by router cache.

3. If in the above steps, a DNS query does not get resolved, then it takes the help of resolver server. Resolver server is nothing but your ISP (Internet service provider). The query is sent to ISP where DNS query runs in ISP cache.

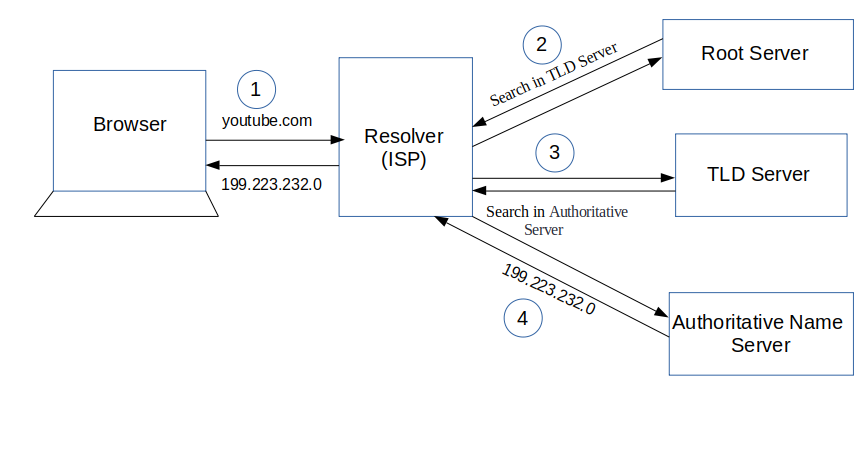
4. If in 3rd steps as well, no results found, then request sends to top or root server of the DNS hierarchy. There it never happens that it says no results found, but actually it tells, from where this information you can get. If you are searching IP address of the top level domain (.com,.net,.Gov,. org). It tells the resolver server to search TLD server (Top level domain).

5. Now, resolver asks TLD server to give IP address of our domain name. TLD stores address information of domain name. It tells the resolver to ask it to Authoritative Name server.

6. The authoritative name server is responsible for knowing everything about the domain name. Finally, resolver (ISP) gets the IP address associated with the domain name and sends it back to the browser.

After getting an IP address, resolver stores it in its cache so that next time, if the same query comes then it does not have to go to all these steps again. It can now provide IP address from their cache.

This is all about the steps that is followed to resolve IP address that is associated with the domain name.



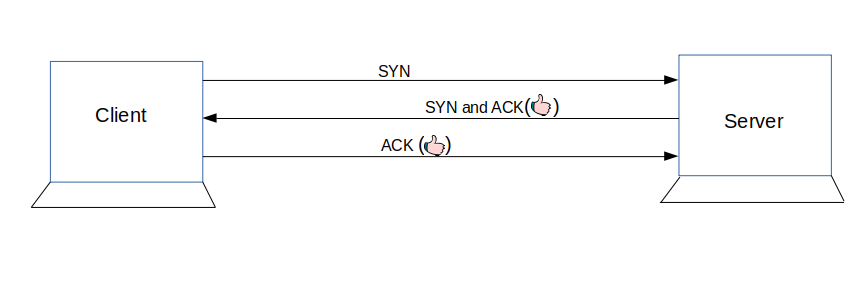
**TCP connection initiates with the server by Browser**

Once the IP address of the computer (where your website information is there) is found, it initiates connection with it. To communicate over the network, internet protocol is followed. TCP/IP is most common protocol. A connection is built between two using a process called ‘TCP 3-way handshake’. Let’s understand the process in brief:

1. A client computer sends a SYN message means, whether second computer is open for new connection or not.

2. Then another computer, if open for new connection, it sends acknowledge message with SYN message as well.

3. After this, first computer receives its message and acknowledge by sending an ACK message.



**Communication Starts (Request Response Process)**

Finally, the connection is built between client and server. Now, they both can communicate with each other and share information. After successful connection, browser (client) sends a request to a server that I want this content. The server knows everything of what response it should send for every request. Hence, the server responds back. This response contains every information that you requested like web page, status-code, cache-control, etc. Now, the browser renders the content that has been requested.