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

**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. In contrast, 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.

**Server push:**

Typically, 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. 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 – like if Bob had sent Alice a Table of Contents of his novel before sending the whole thing.

**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. However, 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.

|  |  |
| --- | --- |
| **HTTP/1.1** | **HTTP/2** |
| It supports connection reuse i.e. for every TCP connection there could be multiple requests and responses, and pipelining where the client can request several resources from the server at once. However, pipelining was hard to implement due to issues such as head-of-line blocking and was not a feasible solution. | Uses multiplexing, where over a single TCP connection resources to be delivered are interleaved and arrive at the client almost at the same time. It is done using streams which can be prioritized, can have dependencies and individual flow control. It also provides a feature called server push that allows the server to send data that the client will need but has not yet requested. |
| HTTP/1.1 provides faster delivery of web pages and reduces web traffic as compared to HTTP/1.0. However, TCP starts slowly and with domain sharding (resources can be downloaded simultaneously by using multiple domains), connection reuse and pipelining, there is an increased risk of network congestion. | HTTP/2 utilizes multiplexing and server push to effectively reduce the page load time by a greater margin along with being less sensitive to network delays. |
| Headers are sent on every request leading to a lot of duplicate data being sent uncompressed across the wire. | Header compression is included by default in HTTP/2 using HPACK. |
| Text based protocol that is in the readable form. | It is a binary protocol (HTTP requests are sent in the form of 0s and 1s). Needs to be converted back from binary in order to read it. |
| SSL is not required but recommended. Digest authentication used in HTTP1.1 is an improvement over HTTP1.0. HTTPS uses SSL/TLS for secure encrypted communication. | Though security is still not mandatory, it is mostly encrypted (though it is not enforced) since almost all clients require traffic to be encrypted. It also has some minimum standards, such as minimum key size for encryption. TLS 1.2 etc. |
| Expands on the caching support by using additional headers like cache-control, conditional headers like If-Match and by using entity tags. | HTTP/2 does not change much in terms of caching. With the server push feature if the client finds the resources are already present in the cache, it can cancel the pushed stream. |
| It is relatively secure since it uses digest authentication, NTLM authentication. | Security concerns from previous versions will continue to be seen in HTTP/2. However, it is better equipped to deal with them due to new TLS features like connection error of type Inadequate\_Security. |

**2. HTTP version history**

The Hypertext Transfer Protocol (HTTP) is one of the most ubiquitous and widely adopted application protocols on the Internet: it is the common language between clients and servers, enabling the modern web.

**HTTP/0.9 (1991)— The One-line Protocol**

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

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

Methods supported: GET only

Response type: hypertext only

Connection nature: terminated immediately after the response

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

**HTTP/1.0 (1996) — Building extensibility**

Browser-friendly protocol

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

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

Methods supported: GET , HEAD , POST

Connection nature: terminated immediately after the response

**HTTP/1.1 (1997)— The standardized protocol**

This is the HTTP version currently in common use.

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.

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

Connection nature: long-lived

**HTTP/2 (2015) — Improving Transport Performance**

It is a binary protocol rather than text. It can no longer be read and created manually. Despite this hurdle, improved optimization techniques can now be implemented.

It is a multiplexed protocol. Parallel requests can be handled over the same connection, removing the order and blocking constraints of the HTTP/1.x protocol.

It compresses headers. As these are often similar among a set of requests, this removes duplication and overhead of data transmitted.

It allows a server to populate data in a client cache, in advance of it being required, through a mechanism called the server push.

Support of Alt-Svc allows the dissociation of the identification and the location of a given resource, allowing for a smarter CDN caching mechanism.

The introduction of Client-Hints allows the browser, or client, to proactively communicate information about its requirements, or hardware constraints, to the server.

The introduction of security-related prefixes in the Cookie header, now helps guarantee a secure cookie has not been altered.

**3. List 5 difference between Browser JS(console) vs Nodejs**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Javascript** | **NodeJS** |
| 1 | Javascript is a programming language that is used for writing scripts on the website. | NodeJS is a Javascript runtime environment. |
| 2 | Javascript can only be run in the browsers. | NodeJS code can be run outside the browser. |
| 3 | It is basically used on the client-side. | It is mostly used on the server-side. |
| 4 | Javascript is capable enough to add HTML and play with the DOM. | Nodejs does not have capability to add HTML tags. |
| 5 | Javascript can run in any browser engine as like JS core in safari and Spidermonkey in Firefox. | Nodejs can only run in V8 engine of google chrome. |
| 6 | Javascript is used in frontend development. | Nodejs is used in server-side development. |
| 7 | Some of the javascript frameworks are RamdaJS, TypedJS, etc. | Some of the Nodejs modules are Lodash, express etc. These modules are to be imported from npm. |

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

**You type maps.google.com into the address bar of your browser.**

**The browser checks the cache for a DNS record to find the corresponding IP address of maps.google.com.**

DNS (Domain Name System) is a database that maintains the name of the website (URL) and the particular IP address it links to. Every single URL on the internet has a unique IP address assigned to it. The IP address belongs to the computer which hosts the server of the website we are requesting to access. DNS is a list of URLs, and their IP addresses, like how a phone book is a list of names and their corresponding phone numbers.

To find the DNS record, the browser checks four caches.

* First, it checks the browser cache. The browser maintains a repository of DNS records for a fixed duration for websites you have previously visited.
* Second, the browser checks the OS cache. If it is not in the browser cache, the browser will make a system call (i.e., gethostname on Windows) to your underlying computer OS to fetch the record since the OS also maintains a cache of DNS records.
* Third, it checks the router cache. If it’s not on your computer, the browser will communicate with the router that maintains its’ own cache of DNS records.
* Fourth, it checks the ISP cache. If all steps fail, the browser will move on to the ISP. Your ISP maintains its’ own DNS server, which includes a cache of DNS records, which the browser would check with the last hope of finding your requested URL.

**If the requested URL is not in the cache, ISP’s DNS server initiates a DNS query to find the IP address of the server that hosts maps.google.com.**

**The browser initiates a TCP connection with the server.**

Once the browser receives the correct IP address, it will build a connection with the server that matches the IP address to transfer information. Browsers use internet protocols to build such connections. There are several different internet protocols that can be used, but TCP is the most common protocol used for many types of HTTP requests.

**The browser sends an HTTP request to the webserver.**

Once the TCP connection is established, it is time to start transferring data The browser will send a GET request asking for maps.google.com web page. If you’re entering credentials or submitting a form, this could be a POST request.

**The server handles the request and sends back a response.**

The server contains a webserver (i.e., Apache, IIS) that receives the request from the browser and passes it to a request handler to read and generate a response. The request handler is a program (written in ASP.NET, PHP, Ruby, etc.) that reads the request, its’ headers, and cookies to check what is being requested and also update the information on the server if needed. Then it will assemble a response in a particular format (JSON, XML, HTML).

**The server sends out an HTTP response.**

The server response contains the web page you requested as well as the status code, compression type (Content-Encoding), how to cache the page (Cache-Control), any cookies to set, privacy information, etc.

**The browser displays the HTML content (for HTML responses, which is the most common).**

The browser displays the HTML content in phases. First, it will render the bare bone HTML skeleton. Then it will check the HTML tags and send out GET requests for additional elements on the web page, such as images, CSS stylesheets, JavaScript files, etc. These static files are cached by the browser, so it doesn’t have to fetch them again the next time you visit the page. In the end, you’ll see maps.google.com appearing on your browser.