**Difference between HTTP1.1 vs HTTP2**

HTTP (Hypertext Transfer Protocol) is indeed the foundation of data communication on the World Wide Web. It governs how information is exchanged between a web client (such as a browser) and a web server. The evolution from HTTP/1.1 to HTTP/2 brought about significant enhancements in performance and efficiency, reflecting the changing demands of internet usage over time.

HTTP facilitates communication between clients and servers on the internet. Users utilize HTTP to send requests to servers, which then respond with the requested data. While HTTP has undergone various stages of development, our focus lies on HTTP/1.1, established in 1997, and its successor, HTTP/2, introduced in 2015

let’s visualize a scenario where you request the homepage of a website, say 'example.com,' and the server responds by delivering the requested HTML page. Now, let's say you navigate to a different section of the site, perhaps to view an image named 'example.jpg.' Despite the sequential nature of these requests, the connection between your browser and the server remains open due to the inclusion of a 'keep-alive' header in your initial request. This persistence allows for multiple requests and responses to be consolidated within a single connection, enhancing efficiency.

However, despite its widespread use, HTTP/1.1 is not without its shortcomings. Firstly, it transmits all requests and responses in plain text, potentially compromising security. Additionally, a phenomenon known as 'head-of-line blocking' occurs, wherein subsequent requests are held up until preceding responses are fully received. Moreover, redundant header information is transmitted with each request, contributing to unnecessary overhead.

To illustrate, let’s consider a scenario involving a popular news website. Upon accessing the homepage, the server delivers the latest headlines and articles. Subsequently, as you click on an article to view its accompanying images, such as those depicting breaking news events, the server responds accordingly. Throughout this interaction, the persistent connection ensures smooth data transfer, yet the inefficiencies inherent in HTTP/1.1 become apparent.

These limitations paved the way for the development of HTTP/2, which aimed to address these issues and propel web communication into a new era of speed and efficiency



HTTP/2 operates on a binary framing layer, converting all messages into binary format for enhanced efficiency. Rather than transmitting data in plain text, HTTP/2 utilizes this binary framing, optimizing the exchange of information between clients and servers. This shift in protocol architecture enables fully multiplexed communication, where a single TCP connection can handle multiple concurrent requests, significantly improving performance.

One of the key features of HTTP/2 is its implementation of HPACK, a header compression algorithm designed to streamline the transmission of header data. By compressing headers, HTTP/2 reduces overhead and minimizes latency, resulting in faster load times and improved user experiences.

Moreover, HTTP/2 introduces server push functionality, allowing servers to proactively send resources such as CSS and JavaScript files to clients without waiting for explicit requests. This preemptive approach enhances page loading times by anticipating and fulfilling the client's needs before they are explicitly stated.

To illustrate the efficacy of HTTP/2, consider a scenario involving an e-commerce platform. Upon accessing the homepage, the server not only delivers the main HTML content but also initiates the transmission of associated CSS stylesheets and JavaScript files through server push. As the user navigates through product listings and adds items to their cart, HTTP/2 seamlessly manages multiple concurrent requests over a single TCP connection, ensuring a smooth and responsive browsing experience.

In summary, HTTP/2 represents a significant leap forward in web communication technology, leveraging binary framing, multiplexing, header compression, and server push to deliver faster, more efficient, and more responsive web experiences.

**Objects and its internal representation in Javascript**

In JavaScript, an object is like a box with labels on it. Each label (or property) points to some information (or value). This information can be anything: a number, a string, another object, or even a function.

Think of objects as big dictionaries where each property (or label) has a corresponding value. When you create an object in JavaScript, behind the scenes, the engine keeps track of these properties and their values using something like a dictionary or a hash table.

To get information from an object, JavaScript looks up the property you're asking for in its internal dictionary. If it finds the property, it gives you its value. If not, it tells you it's undefined, meaning there's no information with that label in the object.

To make things faster, JavaScript engines use tricks like remembering where properties are located or organizing objects in a way that makes finding properties quicker. These tricks help speed up the process of getting information from objects.

Objects can share information with each other through a process called inheritance. If one object doesn't have a certain property, JavaScript looks at its parent object (or prototype) to see if it has it. This allows objects to borrow properties from each other.

We can save objects in a format called JSON (JavaScript Object Notation) using special functions provided by JavaScript. JSON is like a way of writing down all the properties and values of an object in a simple text format. You can later load this text back into JavaScript to recreate the original object.