Differences between HTTP/1.1 and HTTP/2

HTTP Stands for Hyper Text Transfer Protocol, every time when we browse something on web, It is the protocol that both client and server will use to Exchange Data between them.

For Example when a user wants to Access some Website, let's say Economic Times (ET),The Client(our browser) sends a Http Request to the Server Where ET website is Hosted and the server receives the Request and Sends back the Appropriate Http Response to the Client.

The first usable version of Http was created in 1997, we call it as HTTP/1.1.This Version of HTTP is still in use on web. A new version was created in 2015 called HTTP/2.

HTTP/1.1 Vs HTTP/2:

In HTTP/1.0 when we Request the server with HTTP GET Method, the response that we get might not contain a fully rendered page ,instead it contains links for Additional resources that are required for the requested page.

The client discovers that the full rendering of the page requires these additional resources from the server only after it downloads the page

Because of this, the client will have to make additional requests to retrieve these resources.

In HTTP/1.0, the client had to break and remake the TCP connection with every new request, a costly affair in terms of both time and resources.

HTTP/1.1 takes care of this problem by introducing persistent connections and pipelining. With persistent connections, HTTP/1.1 assumes that a TCP connection should be kept open unless directly told to close.

This allows the client to send multiple requests along the same connection without waiting for a response to each, greatly improving the performance of HTTP/1.1 over HTTP/1.0.

Unfortunately, there is a natural bottleneck to this optimization strategy.

Since multiple data packets cannot pass each other when traveling to the same destination, there are situations in which a request at the head of the queue that cannot retrieve its required resource will block all the requests behind it.

This is known as head-of-line (HOL) blocking, and is a significant problem with optimizing connection efficiency in HTTP/1.1.

Adding separate, parallel TCP connections could alleviate this issue, but there are limits to the number of concurrent TCP connections possible between a client and server, and each new connection requires significant resources

PRIORITIZATION

In the context of web performance, prioritization refers to the order in which pieces of content are loaded. Suppose a user visits a news website and navigates to an article.

Should the photo at the top of the article load first? Should the text of the article load first? Should the banner ads load first?

Prioritization affects a webpage's load time.

For example, certain resources, like large JavaScript files, may block the rest of the page from loading if they have to load first. More of the page can load at once if these render-blocking resources load last.

In addition, the order in which these page resources load affects how the user perceives page load time.

If only behind-the-scenes content (like a CSS file) or content the user can't see immediately (like banner ads at the bottom of the page) loads first, the user will think the page is not loading at all.

If the content that's most important to the user loads first, such as the image at the top of the page, then the user will perceive the page as loading faster.

How does prioritization in HTTP/2 affect performance?

HTTP/2 offers a feature called weighted prioritization.

This allows developers to decide which page resources will load first, every time. In HTTP/2, when a client makes a request for a webpage, the server sends several streams of data to the client at once, instead of sending one thing after another.

This method of data delivery is known as multiplexing. Developers can assign each of these data streams a different weighted value, and the value tells the client which data stream to render first.

Imagine that Alice wants to read a novel that her friend Bob wrote, but both Alice and Bob only communicate through the regular mail. Alice sends a letter to Bob and asks Bob to send her his novel. Bob decides to send the novel HTTP/1.1-style: He mails one chapter at a time, and he only mails the next chapter after receiving a reply letter from Alice confirming that she received the previous chapter. Using this method of content delivery, it takes Alice many weeks to read Bob's novel.

Now imagine that Bob decides to send Alice his novel HTTP/2-style: In this case, he sends each chapter of the novel separately (to stay within the postal service's size limits) but all at the same time. He also numbers each chapter: Chapter 1, Chapter 2, etc. Now, Alice receives the novel all at once and can assemble it in the correct order on her own time. If a chapter is missing, she may send a quick reply asking for that specific chapter, but otherwise the process is complete, and Alice can read the novel in just a few days.

In HTTP/2, data is sent all at once, much like Bob when he sends Alice multiple chapters at once. And just like Bob, developers get to number the chapters in HTTP/2. They can decide if the text of a webpage loads first, or the CSS files, or the JavaScript, or whatever they feel is most important for the user experience.

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.