What is HTTP? Why is HTTP/2 faster than HTTP/1.1?

HTTP stands for hypertext transfer protocol, and it is the basis for almost all web applications. More specifically, HTTP is the method computers and servers use to request and send information. For instance, when someone navigates to cloudflare.com on their laptop, their web browser sends an HTTP request to the Cloudflare servers for the content that appears on the page. Then, Cloudflare servers send HTTP responses with the text, images, and formatting that the browser displays to the user.

What is HTTP/1.1?

The first usable version of HTTP was created in 1997. Because it went through several stages of development, this first version of HTTP was called HTTP/1.1. This version is still in use on the web.

What is HTTP/2?

In 2015, a new version of HTTP called HTTP/2 was created. HTTP/2 solves several problems that the creators of HTTP/1.1 did not anticipate. In particular, HTTP/2 is much faster and more efficient than HTTP/1.1. One of the ways in which HTTP/2 is faster is in how it prioritizes content during the loading process.

What is 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?

In HTTP/2, developers have hands-on, detailed control over prioritization. This allows them to maximize perceived and actual page load speed to a degree that was not possible in HTTP/1.1.

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.

What are the other differences between HTTP/2 and HTTP/1.1 that impact performance?

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.

Objects and its internal representation in Javascript

In JavaScript, an object is a standalone entity, with properties and type. Compare it with a cup, for example. A cup is an object, with properties. A cup has a color, a design, weight, a material it is made of, etc. The same way, JavaScript objects can have properties, which define their characteristics.

Creating Objects in JavaScript:

1. By object literal
2. By creating instance of Object directly (using new keyword)

By object literal:

The syntax of creating object using object literal is given below:

```
object={property1:value1,property2:value2.....propertyN:valueN}
```

Property and value is separated by colon(:).

Example:

```
var person={
 fname:"xxx",
 lname:"yyy",
 age: 25
};
```

By creating instance of Object directly (using new keyword):

The syntax of creating object directly is given below:

```
var objectname=new Object();
```

Here, **new keyword** is used to create object.

Example:

```
var emp=new Object();
emp.id=101;
emp.name="xxx";
emp.salary=50000;
```

Accessing JavaScript Objects:

The syntax for accessing the property of an object is:

objectName.property

or

objectName[“*property*”]

Accessing ‘fname’ from example 1 using dot operator,

```
person.fname
```

Accessing ‘name’ form example 2 using [],

```
emp[“name”]
```