**ASSIGNMENT - 1**

**What is HTTP?**

[**HTTP**](https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-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.

A simple and abstract example would be a **restaurant guest and a waiter**. The guest (**Client**) asks (**sends** **request**) waiter (**Server**) for a meal, then the waiter gets the meal from the restaurant chef (**your application logic**) and brings the meal to the guest.

This is a very simplistic example, but it is also the one that will help you understand the concept.

**HTTP/1.1**

Developed by Timothy Berners-Lee in 1989 as a communication standard for the World Wide Web, HTTP is a top-level application protocol that exchanges information between a client computer and a local or remote web server. In this process, a client sends a text-based request to a server by calling a method like GET or POST. In response, the server sends a resource like an HTML page back to the client.

For example, let’s say you are visiting a website at the domain www.example.com. When you navigate to this URL, the web browser on your computer sends an HTTP request in the form of a text-based message, similar to the one shown here:

GET /index.html HTTP/1.1

Host: www.example.com

This request uses the GET method, which asks for data from the host server listed after Host:. In response to this request, the example.com web server returns an HTML page to the requesting client, in addition to any images, stylesheets, or other resources called for in the HTML. Note that not all of the resources are returned to the client in the first call for data. The requests and responses will go back and forth between the server and client until the web browser has received all the resources necessary to render the contents of the HTML page on your screen.

You can think of this exchange of requests and responses as a single application layer of the internet protocol stack, sitting on top of the transfer layer (usually using the Transmission Control Protocol, or TCP) and networking layers (using the Internet Protocol, or IP):

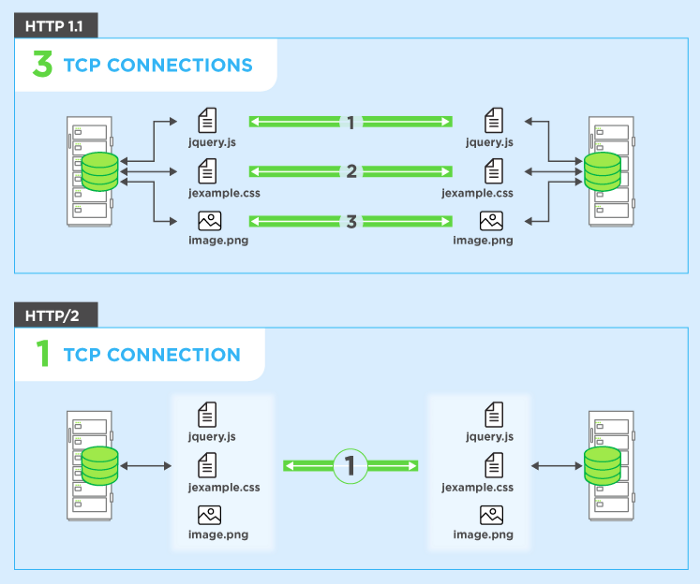
**HTTP/2**

HTTP/2 began as the SPDY protocol, developed primarily at Google with the intention of reducing web page load latency by using techniques such as compression, multiplexing, and prioritization. This protocol served as a template for HTTP/2 when the Hypertext Transfer Protocol working group httpbis of the [IETF (Internet Engineering Task Force)](https://www.ietf.org/) put the standard together, culminating in the publication of HTTP/2 in May 2015. From the beginning, many browsers supported this standardization effort, including Chrome, Opera, Internet Explorer, and Safari. Due in part to this browser support, there has been a significant adoption rate of the protocol since 2015, with especially high rates among new sites.

**Differences Between HTTP/1.1 and HTTP/2.0**

**1. Request multiplexing**

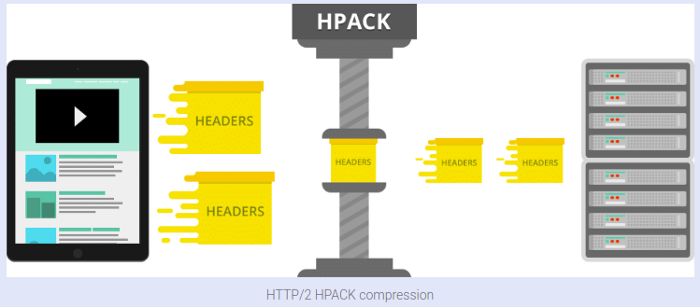
HTTP/2 can send **multiple requests** for data in parallel over a **single** TCP connection. This is **the most** **advanced** **feature** of the HTTP/2 protocol because it **allows you to download web files asynchronously from one server**. Most modern browsers limit TCP connections to one server.



This reduces additional round trip time (RTT), **making your website load faster** without any optimization, and makes domain sharding unnecessary.

**2.Header compression**

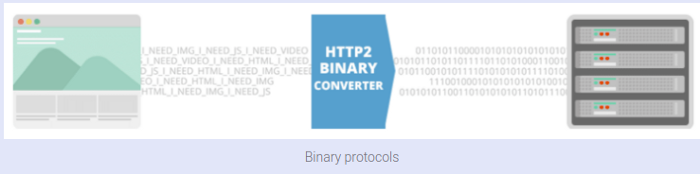
Small files load more quickly than large ones. To speed up web performance, both HTTP/1.1 and HTTP/2 compress HTTP messages to make them smaller. However, HTTP/2 uses a more advanced compression method called HPACK that eliminates redundant information in HTTP header packets. This eliminates a few bytes 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.



**3. Binary protocol**

The latest HTTP version has evolved significantly in terms of capabilities and attributes such as transforming from a text protocol to a binary protocol. HTTP1.x used to process text commands to complete request-response cycles. HTTP/2 will use binary commands (in 1s and 0s) to execute the same tasks. This attribute eases complications with framing and simplifies implementation of commands that were confusingly intermixed due to commands containing text and optional spaces.

Browsers using HTTP/2 implementation will convert the same text commands into binary before transmitting it over the network.

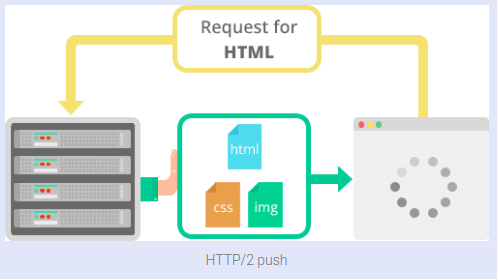


**Benefits:**

* Low overhead in parsing data — a critical value proposition in HTTP/2 vs HTTP1.
* Less prone to errors.
* Lighter network footprint.
* Effective network resource utilization.
* Eliminating security concerns associated with the textual nature of HTTP1.x such as response splitting attacks.
* Enables other capabilities of the HTTP/2 including compression, multiplexing, prioritization, flow control and effective handling of TLS.
* Compact representation of commands for easier processing and implementation.
* Efficient and robust in terms of processing of data between client and server.
* Reduced network latency and improved throughput.

**4. HTTP Server Push**

In HTTP/1.1 This capability allows the server to send only cacheable information to the client that requested, In HTTP/2.0 This capability allows the server to send additional cacheable information to the client that isn’t requested but is anticipated in future requests. For example, if the client requests for the resource X and it is understood that the resource Y is referenced with the requested file, the server can choose to push Y along with X instead of waiting for an appropriate client request.



**Benefits:**

* The client saves pushed resources in the cache.
* The client can reuse these cached resources across different pages.
* The server can multiplex pushed resources along with originally requested information within the same TCP connection.
* The server can prioritize pushed resources — a key performance differentiator in HTTP/2 vs HTTP1.
* The client can decline pushed resources to maintain an effective repository of cached resources or disable Server Push entirely.
* The client can also limit the number of pushed streams multiplexed concurrently.

If you remember the story about a guest in a restaurant and waiter, that would be an example

For HTTP/1.1 and HTTP/2 protocol with a slight difference. Imagine that waiters are TCP connections and you want to order your meal and a bottle of water. For HTTP/1.1 that would mean that you ask one waiter for your meal and another one for water, hence you would allocate two TCP connections. For HTTP/2 that would mean that you ask only one waiter for both, but he brings them separately. You only allocate one TCP connection and that will already result with lower server load, plus the server would have one extra free connection (waiter) for the next client (guest).

The real difference between HTTP/1.1 and HTTP/2 comes with server push example.

Imagine that the guest (Client) asks (sends request) waiter (Server) for a meal, then the waiter gets the meal from the restaurant chef (your application logic), but the waiter also thinks you would need a bottle of water so he brings that too with your meal. The end result of this would be only one TCP connection and only one request that will significantly lower the server load.

What we have here is a simple page with 100 images of checks which I’ll use to demonstrate HTTP/1.1, HTTP/2 AND HTTP/2 server push.

What is important to note in the picture above are number of requests, load time, protocol column, initiator column and waterfall diagram itself (we can see how requests are made through multiple batches, unfortunately, it is hard to see other data from it except TTFB and content download time; eg. resource scheduling and connection start time).

**5.Prioritization**

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](https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/) 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