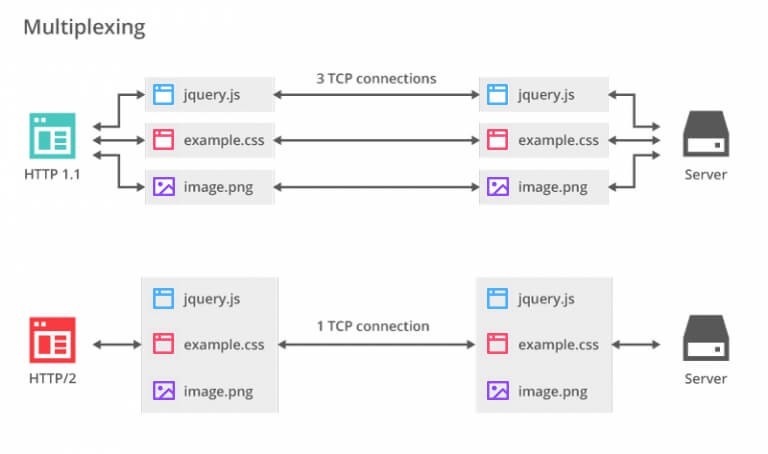
**Difference between HTTP1.1 and HTTP2**

**Single connection and multiplexing**

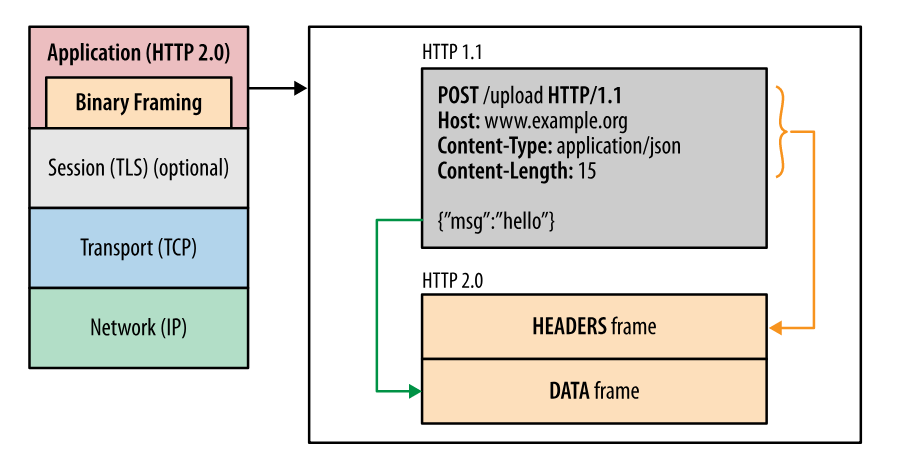
HTTP/1.1 is a sequential protocol, where the browser opens only one TCP connection, requests one file, and only after receiving the file it moves to the next one. If a file is too large, or the server processing is too slow, the page can crash. To minimize this issue, browsers usually open multiple connections, between 6 and 8, per server. In HTTP/2, on the other hand, a TCP connection will be persistent and only one per source is needed since parallel requests and responses can request/receive all the necessary files.

This brings a reduction in processing and memory consumption, a reduction in network operating cost, and increased usability. The result is reduced network latency and lower hardware and software costs.



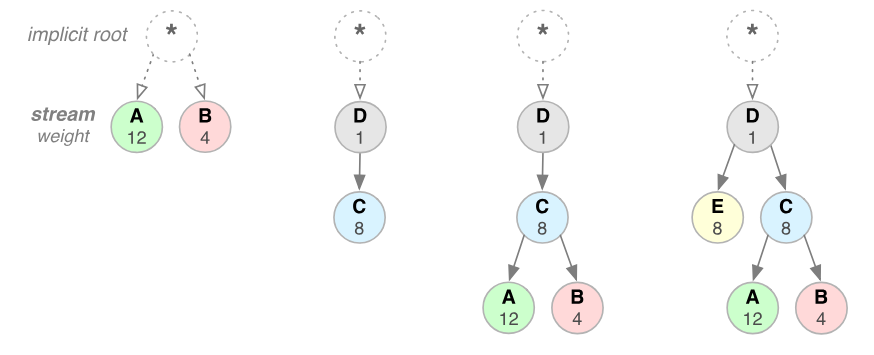
**Binary framing layer**

This is the core of all HTTP/2 performance improvements, determining how HTTP messages are encapsulated and transferred between client and server. The encoding mechanism has been redesigned without changing the semantics of methods, verbs, and headers. Communication is broken into frames, over a single TCP connection.



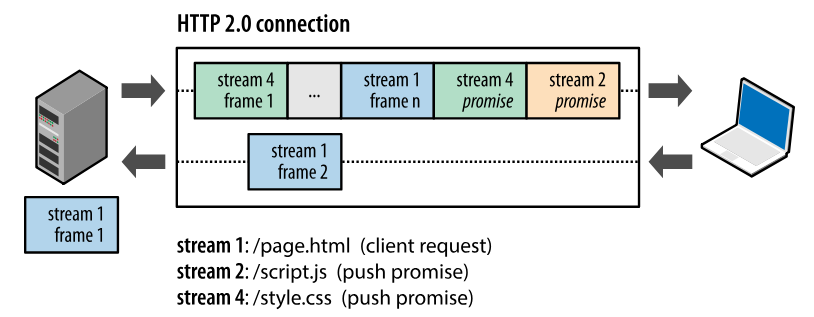
**Request Prioritization**

It enables the browser to request all elements when discovered, communicating to the server its intention to prioritize any of them. This is done through dependencies and stream weights. An example of this is when CSS files and JS files are requested, that with HTTP/2 the browser will prioritize CSS files first, even if its request in DOM order comes after the JS.



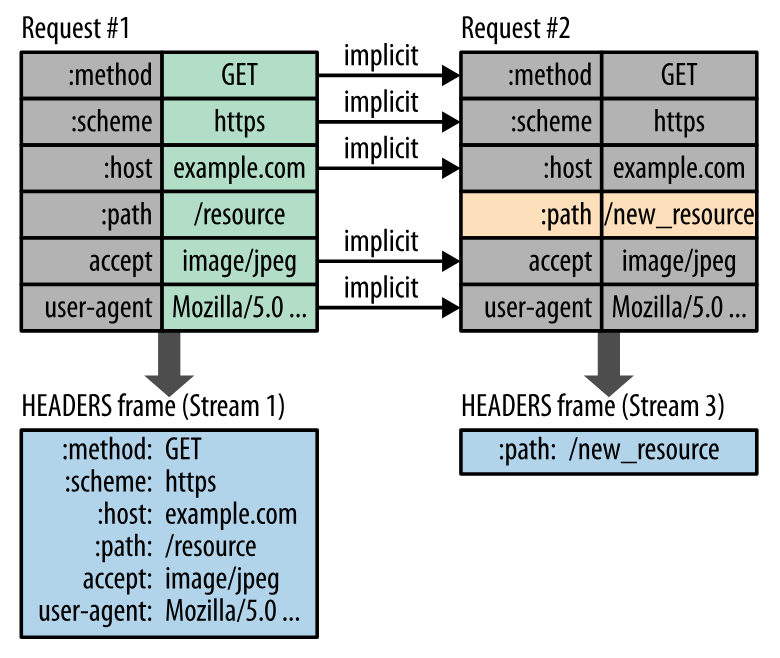
**Server Push**

It allows the server to send multiple responses to the client, from a single request, without the client explicitly asking for it. Imagine the following scenario: a web browser requests a page’s HTML file; the server then responds with the requested file and also sends the CSS file, JavaScript, icons, and other things.



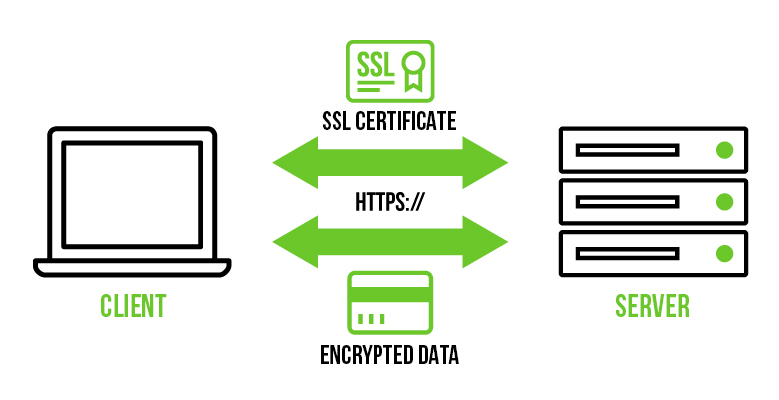
**Automatic compression**

HTTP requests have headers with important information about the resource and its properties. With HTTP/2 the headers are compressed using the HPACK algorithm, thus reducing the size of each transfer and maintaining and updating an indexed list of the header fields seen earlier. In addition, data compression via GZIP, which needs to be enabled in HTTP/1.1, became standard in version 2.



**Data security and encryption**

The HTTP/2 protocol has been implemented to work with or without encryption. However, all major browsers declared that they will only support HTTP/2 with encryption, requiring the use of an SSL certificate.



[IMAGEM: Client, SSL Certification, Encrypted Data, Server]

**Objects and its internal representation in Javascript :**

JavaScript objects are similar to objects in real life which consists of different attributes and properties. These objects are defined as an unordered collection of related data, that are of primitive or reference types. These are defined in the form of “key: value” pairs.

Objects, in JavaScript, is it’s most important data-type and forms the building blocks for modern JavaScript. These objects are quite different from JavaScript’s primitive data-types(Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).

Objects are more complex and each object may contain any combination of these primitive data-types as well as reference data-types.  
An object, is a reference data type. Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the location in memory where the object is stored. The variables don’t actually store the value.

Loosely speaking, objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of “key: value” pairs. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.

For Eg. If your object is a student, it will have properties like name, age, address, id, etc and methods like updateAddress, updateNam, etc.

# ****Objects and properties****

A JavaScript object has properties associated with it. A property of an object can be explained as a variable that is attached to the object. Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects. The properties of an object define the characteristics of the object. You access the properties of an object with a simple dot-notation:

objectName.propertyName

Like all JavaScript variables, both the object name (which could be a normal variable) and property name are case sensitive. You can define a property by assigning it a value. For example, let’s create an object named myCar and give it properties named make, model, and year as follows:

var myCar = new Object();  
myCar.make = 'Ford';  
myCar.model = 'Mustang';  
myCar.year = 1969;

**IP address, port, HTTP methods, MAC address:**

**IP address:**

An Internet Protocol (IP) address is a unique numerical identifier for every device or network that connects to the internet. Typically assigned by an internet service provider (ISP), an IP address is an online device address used for communicating across the internet.

**Port address:**

Port is an address of a 16-bit unsigned integer number which ranges from 0 to 65535. The primary application of a port number is to transmit the data between a Computer Network and an Application. Port is just a unique number assigned to every application of a computer.

**Mac address:**

A MAC address (media access control address) is a 12-digit hexadecimal number assigned to each device connected to the network. Primarily specified as a unique identifier during device manufacturing, the MAC address is often found on a device's network interface card (NIC).

**HTTP methods:**

|  |  |
| --- | --- |
| 1 | **GET**  The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data. |
| 2 | **HEAD**  Same as GET, but transfers the status line and header section only. |
| 3 | **POST**  A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms. |
| 4 | **PUT**  Replaces all current representations of the target resource with the uploaded content. |
| 5 | **DELETE**  Removes all current representations of the target resource given by a URI. |
| 6 | **CONNECT**  Establishes a tunnel to the server identified by a given URI. |
| 7 | **OPTIONS**  Describes the communication options for the target resource. |
| 8 | **TRACE**  Performs a message loop-back test along the path to the target resource. |