**1 . DIFFERENCE BETWEEN HTTP1.1 VS HTTP2**

**HTTP1.1**

In HTTP1.1, flow control relies on the underlying TCP connection. When this connection initiates, both client and server establish their buffer sizes using their system default settings.

If the receiver’s buffer is partially filled with data, it will tell the sender its receive window, i.e., the amount of available space that remains in its buffer.

This receive window is advertised in a signal known as an ACK packet, which is the data packet that the receiver sends to acknowledge that it received the opening signal.

If this advertised receive window size is zero, the sender will send no more data until the client clears its internal buffer and then requests to resume data transmission.

It is important to note here that using receive windows based on the underlying TCP connection can only implement flow control on either end of the connection.

Because HTTP1.1 relies on the transport layer to avoid buffer overflow, each new TCP connection requires a separate flow control mechanism.

HTTP2, however, multiplexes streams within a single TCP connection, and will have to implement flow control in a different manner.

**HTTP2**

HTTP2 multiplexes streams of data within a single TCP connection.

As a result, receive windows on the level of the TCP connection are not sufficient to regulate the delivery of individual streams.

HTTP2 solves this problem by allowing the client and server to implement their own flow controls, rather than relying on the transport layer.

The application layer communicates the available buffer space, allowing the client and server to set the receive window on the level of the multiplexed streams.

This fine-scale flow control can be modified or maintained after the initial connection via a WINDOW\_UPDATE frame.

Since this method controls data flow on the level of the application layer, the flow control mechanism does not have to wait for a signal to reach its ultimate destination before adjusting the receive window.

Intermediary nodes can use the flow control settings information to determine their own resource allocations and modify accordingly. In this way, each intermediary server can implement its own custom resource strategy, allowing for greater connection efficiency.

This flexibility in flow control can be advantageous when creating appropriate resource strategies.

For example, the client may fetch the first scan of an image, display it to the user, and allow the user to preview it while fetching more critical resources. Once the client fetches these critical resources, the browser will resume the retrieval of the remaining part of the image.

|  |  |  |
| --- | --- | --- |
|  | HTTP1.1 | HTTP2 |
| YEAR | 1997 | 2015 |
| KEY FEATURES | It supports connection reuse i.e. for every TCP connection there could be multiple requests and responses, and pipelining where the client can request several resources from the server at once. However, pipelining was hard to implement due to issues such as head-of-line blocking and was not a feasible solution. | Uses multiplexing, where over a single TCP connection resources to be delivered are interleaved and arrive at the client almost at the same time. It is done using streams which can be prioritized, can have dependencies and individual flow control. It also provides a feature called server push that allows the server to send data that the client will need but has not yet requested. |
| STATUS CODE | Introduces a warning header field to carry additional information about the status of a message. Can define 24 status codes, error reporting is quicker and more efficient. | Underlying semantics of HTTP such as headers, status codes remains the same. |
| AUTHENTICATION MECHANISM | It is relatively secure since it uses digest authentication, NTLM authentication. | Security concerns from previous versions will continue to be seen in HTTP/2. However, it is better equipped to deal with them due to new TLS features like connection error of type Inadequate\_Security. |
| CACHING | Expands on the caching support by using additional headers like cache-control, conditional headers like If-Match and by using entity tags. | HTTP/2 does not change much in terms of caching. With the server push feature if the client finds the resources are already present in the cache, it can cancel the pushed stream. |
| WEB TRAFFIC | HTTP/1.1 provides faster delivery of web pages and reduces web traffic as compared to HTTP/1.0. However, TCP starts slowly and with domain sharding (resources can be downloaded simultaneously by using multiple domains), connection reuse and pipelining, there is an increased risk of network congestion | HTTP/2 utilizes multiplexing and server push to effectively reduce the page load time by a greater margin along with being less sensitive to network delays. |

**2. OBJECTS AND ITS INTERNAL REPRESENTATION IN JAVASCRIPT**

* Objects are important data types in javascript.
* Objects are different than primitive datatypes (i.e. number, string, boolean, etc.).
* Primitive data types contain one value but Objects can hold many values in form of Key: value pair.
* These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.
* Every object has some property associated with some value. These values can be accessed using these properties associated with them.A JavaScript object is a collection of named values having state and behavior.
* Examples: person, car, bike, computer, washing machine etc..,, Take the case of cars.
* All cars have the same properties, but the property values differ from car to car. All cars have the same methods, but the methods are performed at different times.
* Let’s have an example of Mercedes car and list out its properties.

Make: Mercedes

Model: C-Class

Color: White

Fuel: Diesel

Weight: 850kg

Mileage: 8Kmpl

Rating: 4.5

* var car = "Mercedes";
* Objects are variables too. But objects can contain many values.
* The following code assigns many values (Mercedes, C-class, White and soo on) to a variable named Car.

var car = {Make: “Mercedes”, Model: “C-Class”, Color: “White”, Fuel: Diesel, Weight: “850kg”, Mileage: “8Kmpl”, Rating: 4.5};

* The values are written as name: value pairs.
* So, conclusion and definition for JS objects is “JavaScript objects are containers for named values”.
* The name:values pairs (in JavaScript objects) are called **properties**.
* Example: Property- Make, property value- Mercedes