**Difference between HTTP1.1 and HTTP2**

1. Delivering model

HTTP1.1 – sends all the messages in plain text format. It uses a persistent connection between two client and the server. Until one connection is explicitly closed, this connection remains open.

But this approach ensures good performance, but there is one disadvantage is that if a request queue cannot retrieve its required resources, then it can block all the request behind it, which is called as ‘Head-of-line-blocking’ (HOL), which is resolved in HTTP2.

HTTP2 - uses a binary framing layer to encapsulate all messages in the binary format. It establishes a single connection object between client and server. It will partition request and response in tiny data packets and encode them. Due to this multiple request and response can be run parallelly.

1. Stream Prioritization

In HTTP1.1, multiple streams awaiting for the same resources can still causes performance issue. This can resolved using the “Stream Prioritization” in HTTP2. When a client sends concurrent requests to a server, it can prioritize the responses it is requesting by assigning a weight between 1 and 256 to each stream. The higher number indicates higher priority. Programmers can enjoy better control on page rendering process with stream prioritization ability.

1. Buffer flow

Server and client machine TCP connection requires both of these to have a certain buffer space for holding incoming requests. Though these buffers can hold numerous or large requests, they may also lack space due to small or limited buffer size. It causes buffer overflow at receiver’s end, resulting in data packet loss.

HTTP1.1 - The flow control mechanism in HTTP/1.1 relies on the basic TCP connection. In beginning itself, both the machines set their buffer sizes automatically. If the receiver’s buffer is full, it shares the receive window details, telling how much available space is left. The receiver acknowledges the same and sends an opening signal.

Note : that flow control can only be implemented on either end of the connection. Moreover, since HTTP/1.1 uses a TCP connection, each connection demands an individual flow control mechanism.

HTTP2 - It multiplexes data streams utilizing the same (one) TCP connection. So, in this case, both machines can implement their flow controls instead of using the transport layer. The application layer shares the available buffer size data, after which, both machines set their receive window details on the multiplexed streams level. In addition, the flow control mechanism does not need to wait for the signal to reach its destination before modifying the receive window.

Compression

Every HTTP transfer contains headers that describe the sent resource and its properties. This metadata can add up to 1KB or more of overhead per transfer, impacting the overall performance. For minimizing this overhead and boosting performance, compressions algorithms must be used to reduce the size of HTTP messages that travels between the machines.

HTTP1.1 - HTTP/1.x uses formats like gzip to compress the data transferred in the messages. However, the header component of the message is always sent as plain text. Though the header itself is small, it gets larger due to the use of cookies or an increased number of requests.

HTTP 2 - To deal with this bottleneck, HTTP/2 uses HPACK compression to decrease the average size of the header. This compression program encodes the header metadata using Huffman coding, which significantly reduces its size as a result. In addition, HPACK keeps track of previously transferred header values and further compresses them as per a dynamically modified index shared between client and server.

4. Predicting Resource Requests

The client receives an HTML page on sending a GET request. While examining the page contents, the client determines that it needs additional resources for rendering the page and makes further requests to fetch these resources. As a consequence of these requests, the connection load time increases. Since the server already knows that the client needs additional files, it can save the client time by sending these resources before requesting; thus, offering a great solution to the problem.

HTTP1.1 - To accomplish this, HTTP/1.1 has a different technique called resource inlining, wherein the server includes the required source within the HTML page in response to the initial GET request. Though this technique reduces the number of requests that the client must send, the larger, non-text format files increase the size of the page.

As a result, the connection speed decreases, and the primary benefit obtained from it also nullifies. Another drawback is the client cannot separate the inlined resources from the HTML page. For this, a deeper level of control is required for connection optimization – a need that HTTP/2 meets with server push.

HTTP2 - As HTTP/2 supports multiple simultaneous responses to the client’s initial GET request, the server provides the required resource along with the requested HTML page. This is called the server push process, which performs the resource inlining like its precursor while keeping the page and the pushed resource separate. This process fixes the main drawback of resource inlining by enabling the client machine to decide to cache/decline the pushed resource separate from the HTML page.