HTTP/2 Streams: Is the Future of WebSockets decided?

Max Fietze

Faculty of Computer Science, TU Dresden

Abstract. The communication in the World Wide Web is mainly realized by the HTTP protocol. First standardized in 1996 it was primarily developed for the transmission of pure text-based websites. Over the years the websites became more and more complex and now contain interactive media, which results in a very inefficient transmission of the data. Because of the fact that the HTTP protocol remained unchanged for more than a decade, the WebSockets standard was developed for an optimized communication. Nearly 20 years after the first HTTP protocol, a new version, named HTTP/2, was published with many enhancements. This paper discusses whether HTTP/2 will make WebSockets obsolete in the future. For a better comprehension, a brief overview of HTTP/1.0 and HTTP/1.1 will be presented including a discussion of deficits. After that a short summary what is new in HTTP/2 and what WebSockets characterizes will be given. Finally these two protocols with their benefits or disadvantages will be compared and then the paper gives an assessment if one of the protocols is able to replace the other.

Keywords: HTTP, WebSockets, Comparison

1 History of HTTP

The first attempts to develop a standard for transferring webpages over a shared network started at the CERN Research Center, mainly under the direction of Roy Fielding and Tim Berners-Lee in 1989. A first basic approach included the GET request and described a simple client-server communication. This "Original HTTP" was published in 1991, and is known as HTTP 0.9 [4].

1.1 HTTP/1.0

HTTP/1.0 added many features to the original HTTP protocol, such as more request methods (HEAD, POST, e.t.c.) but still relied on the restriction that every resource request has to be made over a separate connection [5]. This implied a big overhead when a website consisted of many different elements, such as pictures.

1.2 HTTP/1.1

In 1999, the HTTP/1.1 standard was published and introduced the additional Keep-Alive header, which instructs the server to keep the connection for further client requests. This allows sending of multiple requests over a single TCP connection, without closing the connection after each request (see figure 1) and also the "pipelining" of requests. Pipelining means that a client can send multiple requests without waiting for the corresponding responses [6], which can decrease the load time especially over a connection with a high Round Trip Time (RTT). In figure 2 you can see a comparison between a connection without and with pipelining.

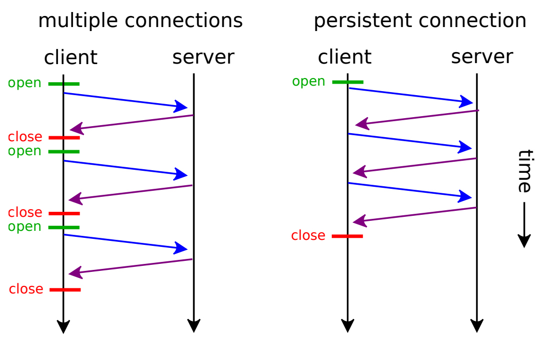

Fig. 1. Comparison between multiple connections and a single persistent connection [25]

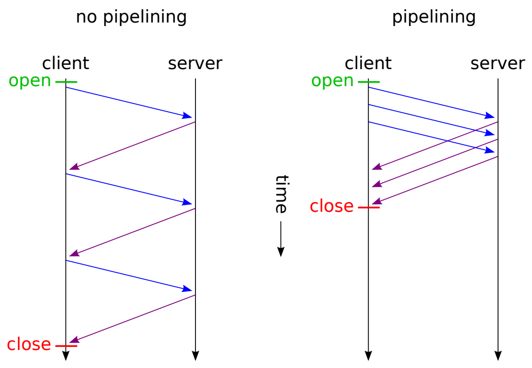

Fig. 2. Pipelining of requests [26]

2 Key features of HTTP/2

HTTP/2 is just a revision of the older HTTP/1.1 Protocol. Therefore, it supports all of the core features of HTTP/1.1, however, tries to be more efficient in many ways [1]. It was mainly inspired by the experimental SPDY protocol, developed by Google[23]. Different to the past version numbers, only the whole integer is mentioned, so it is only called HTTP/2 instead of HTTP/2.0. In the following section we will summarize some of the key features, that were added or semantics that are used differently:

Streams and Multiplexing: A very common problem of HTTP/1.1 is the so called "Head-of-line blocking" (HOL blocking), which can dramatically decrease the performance of the page load time [2]. This phenomenon occurs because the requests over a pipelined TCP connection are processed in the FIFO principle (First in, first out), so a single slow request can suspend all other. HTTP/2 uses streams consisting of single frames. Each frame contains an stream ID, which is used to reassign the frames to the corresponding stream. This allows the Client and Server to send many different streams at the same time over a single TCP connection (called Multiplexing, see figure 3), which is an fundamental difference to the HTTP/1.1 delivery model. Also it avoids the HOL problem, because when one streams stucks, another stream can continue sending. This increases the usage of available bandwidth and makes workarounds like spriting of images unnecessary.

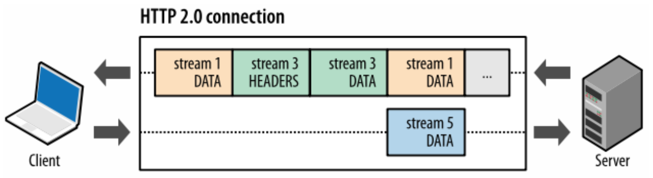

Fig. 3. Multiple different streams within a shared connection [27]

Stream Priority and Dependencies: A possible disadvantage of a multiplexed connection can be the interdependence of the streams, especially when a stream needs information of a so far not transferred stream.This can delay for example an interim rending of a website before it is fully loaded. To resolve this problem, each stream can be assigned to a specific priority or information about interdependencies, which is defined in the header information [9]. This should be minded when processing the streams, especially when the capacity for sending is limited.

Header field compression: HTTP/1.1 did not had any standardized header compression algorithms, although the headers of HTTP requests are highly redundant and uniformly. For a more effective communication and to reduce the data size of the headers HTTP/2 introduced the HPACK algorithm [3]. It uses a header table which contains 61 static table entries with very common and often used headers and a dynamic table which keeps track of used header parts and caches them with an identifier. When a header is reused, the sender just has to send the corresponding identifier, which decreases the amount of data that has to be sent dramatically.

Binary protocol: While HTTP/1.1 was still a text-based protocol, HTTP/2 is full binary. The advantage is that binary data is much more efficient to handle and the coding for transmission is more efficient, too. Furthermore, there is an well-defined standard how the data has to be parsed, while there were overall four different ways under HTTP/1.1, which was very likely to cause errors [7] [8].

Server-Push: Under HTTP/1.1 the client was fully responsible to extract the required requests from the initial response for a complete rendering of a website, like style sheets, and to request these additional resources themselve. With HTTP/2 the server is able to proactively promise the client which resources it is very likely to need and send them in the cache self-initiated, before the client requests them. When the client recognizes which resources it needs, but already received them by the server-push, it don't has to request them again from the server. This saves time and decreases the amount of requests in the network.

3 Introduction to WebSockets

Web applications were steadily developed following the request-response system, which means that the communication was always initiated by the client to the server, requesting to send new data. But especially modern real-time applications require, that the server can send the client self-initiated data at certain events, even when the client doesn't explicitly requests them. A very common way to achieve this over HTTP is the so-called "Long polling", which uses the traditional polling technique, but the client is not expecting an immediate response and keeps the connection open till the server sends a response when an event occurs. The biggest issue of this technique is that every new request has to use a complete header, which causes a very high overhead (compared to WebSockets see section 4). Additionally, the client has to send a request after every response from the server, but in the meantime between the response from the server and the new request, the server is not able to respond to the client, which can delay notifications from the server. This makes applications that need a very low latency nearly impossible.

A widespread approach to solve the problems and allow an efficient bidirectional communciation is the WebSocket technique. It is a TCP-based protocol, which was standardized by the IETF in RFC 6455. In contrast to HTTP it offers a real bidirectional communication, without constant polling by the client. It consists of two parts: a handshake and the data transfer [16]. The handshake

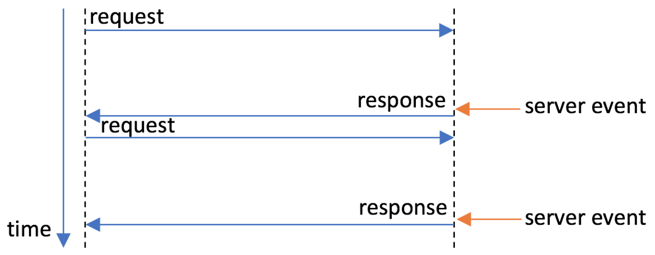

Fig. 4. Structure of a communcation with long polling

executes a HTTP request, which contains a defined HTTP Upgrade request [17]. After the handshake the data transfer uses own headers, which are limited to the most necessary information and cause just a small overhead. The opportunity of bidirectional connection allows manifold applications, like real-time social feeds, multiplayer games or collaborative editing of documents, which were much more difficult with the HTTP protocol, due to the high latency. WebSockets are currently supported by all major web browsers.

4 Comparison between HTTP/2 and WebSockets

WebSockets is an independent TCP-based protocol, which only relationship to HTTP is the handshake to upgrade the connection [11]. Therefore, a comparison between these two techniques is basically possible, but on different levels. Both techniques offer a better bandwidth usage through multiplexing and a more efficient processing through the binary representation (optional for WebSockets). HTTP/2 offers additionally a very efficient compression algorithm for the headers. However, compared to the uncompressed headers of WebSockets this is not a huge benefit, because the headers of WebSockets are extremely small from scratch. When it comes to the ability of a real bidirectional communication, both protocols start to differentiate. Although HTTP/2 differs from the past HTTP protocols that the server is able to send data self-initiated, this is still only possible if the client sent an initial request before. This means the client has to initiate a new request when it expects new data from the server. Compared to WebSockets, this is not a full bidirectional communication. Although it is basically possible to allow a bidirectional communication with HTTP/2 (e.g., with long polling or server sent events), its not designed for this purpose. So there still has to be sent more requests from the client to the server, because the server is not able to initiate the communication at its own, and the client has to send requests even if no update from the server is available. The only advantage of using HTTP/2 instead of HTTP/1.1 however is, that the size of the requests are much smaller causing less traffic in the network. WebSockets are designed from scratch for an efficient bidirectional communication and is accordingly optimized, therefore the overhead for communication is extremely low. When it comes to scalability of applications using bidirectional communication WebSockets have an real advantage compared to HTTP. Considering the overhead of a HTTP communication, an average HTTP/1.1 header has a size of around 700-800 bytes [20] [21]. A test of KeyCDN resulted that the header size of HTTP/2 is decreased about 30% compared to HTTP/1.1 [24], which means that an average HTTP/2 header has the size of about 550 bytes (rounded). Compared to a typical WebSocket header which has a size of 2 Bytes [21] it is a huge difference, especially when considering many clients communicating with a server. For that we created an interpolation with 3 different use cases considering the pure header overhead (inspired by [21]), on the one side a specified amount of clients using normal HTTP polling and on the other side the clients are using WebSockets (see table 1). When illustrating the results graphically the difference is obvious (see figure 5), the header ratio between HTTP polling and WebSockets is about 275:1. Assuming there are 1 million clients requesting the server every second, this means that HTTP polling causes an overhead of 4,4 Gbit/s just for the headers. WebSockets cause in contrast just 16 Mbit/s of overhead. Another difference that has to be kept in mind for further consideration is the year of introduction. While the WebSockets protocol was standardized in 2011, HTTP/2 is a relatively young protocol and standardized just in 2015. For a better overview about the similarities and differences take a look at table 2.

Table 1. Interpolation about bandwidth usage of HTTP and WebSockets (4400 bits vs. 16 bits header) by header overhead, Overhead in megabit per second

	Number of clients	Overhead HTTP	Overhead WebSockets
Use Case A	10000	44	0,16
Use Case B	100000	440	1,6
Use Case C	1000000	4400	16

Table 2. Tabular comparison betwenn HTTP/2 and WebSockets

	HTTP/2	WebSockets
Type of representation	Binary	Binary or textual
Header Compression	Yes	No
Multiplexing	Yes	Yes
Priorization of Streams	Yes	No
Direction	Client to Server and Server-Push	Bidirectional
Scalability	Poor	Good
Year of Introduction	2015	2011

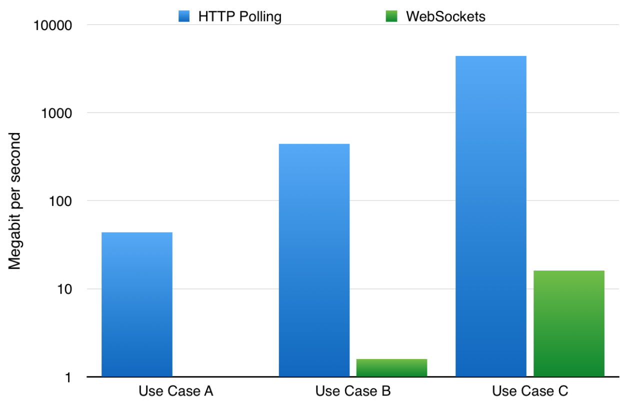

Fig. 5. Graphical representation of table 1, consider logarithmic scale

5 Current Adoption Rates

To get an overview, which protocol will prevail against the other over time, it is useful to take a look at the current adoption rates. On the Internet are various websites which analyze the most popular websites according to the used techniques.

For the consideration auf HTTP/2 this paper refers primarily to the data available from W3Techs[1], which analyzes the top 10 million websites (unique addresses, no subdomains). The popularity ranking of the analyzed websites is provided by the data of Alexa, an Amazon.com company [12]. The statistic is reported and published daily. In figure 6 the current data as from May 23, 2017 can be seen. The visible data reaches back one year, so till May 2016, approximately one year after the official standardization of HTTP/2. A clear upward tendency, which rises nearly linear can be seen. On May 23, 2017, the current spread of HTTP/2 was 14,2%, based on the analyzed websites. Considering the fact that HTTP/2 is still a very young protocol, the previous distribution is quite remarkable.

Another interesting statistic can be found on HTTP/2 Dashboard[2], which is specialized in the adoption of HTTP/2. The site, unlike W3Techs, analyzes each subdomain individually and then subdivides the web pages according to whether they have announced the use of HTTP/2, only partial support or if they support HTTP/2 fully. Note the ascents, marked by the red lines 1 and 3. At the time of 1 Cloudflare implemented the support, whereby nearly all web pages, which use the Cloudflare proxy technology now also supports HTTP/2 [13]. The same

[1] `www.w3techs.com/technologies/details/ce-http2/all/all` – Retrieved May 23, 2017

[2] `http://isthewebhttp2yet.com/measurements/adoption.html` – Retrieved May 23, 2017

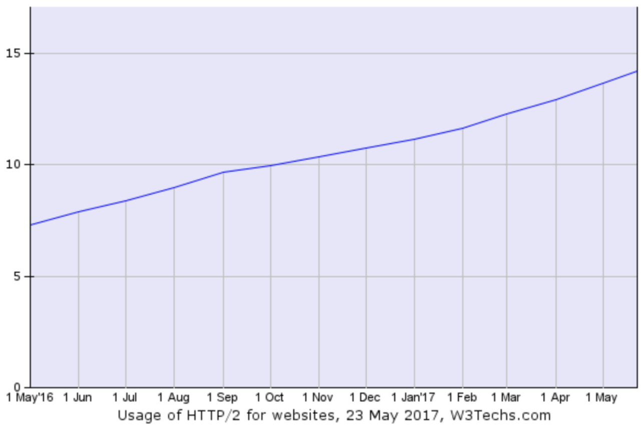

Fig. 6. Historical trend of the adoption of HTTP/2 since May 2016 from W3Techs [28]

applies to point 3, where wikipedia and blogspot enabled the use of HTTP/2 [14] [15]. It can be seen that large content providers are moving their infrastructure to HTTP/2, which gives the spread of HTTP/2 a big boost. Notice that the rise at line 2 is just because of an adapted measurement methodology. In conclusion, the spread of HTTP/2 is evolving steadily and it is assumed that it will be the dominant HTTP standard in the near future.

Regarding the adoption of WebSockets, it is more difficult to obtain reliable data. This is probably due to the fact that the use of WebSockets can not be determined directly by a normal page call since WebSocket applications are usually started by a user interaction or because the adoption is comparatively low to other techniques (see further below). Fortunately many current implementations use the Socket.IO framework, which uses mainly WebSockets and polling as a fallback option [18]. Socket.IO is monitored by the website of BuiltWith[3] and offers statistics like the two mentioned before. A current trend diagram can be found in figure 8, where a relatively constant behavior with a slight trend downwards can be seen. The adoption among the top 10000 websites (according to Quantcast[4]) is just about 0,2% [19]. However, since it is only the specific use of the Socket.IO API it is not possible to extract an exact value how many websites use WebSockets on an average. But the very low adoption among the top 10000 websites of the very popular Socket.IO API suggests, that the technique of WebSockets is not very accepted so far. Whether it will change in the near future is hard to say, but compared to HTTP/2, WebSockets are almost meaningless.

[3] `www.trends.builtwith.com/javascript/Socket.IO` – Retrieved May 23, 2017
[4] `www.quantcast.com` – Retrieved May 23, 2017

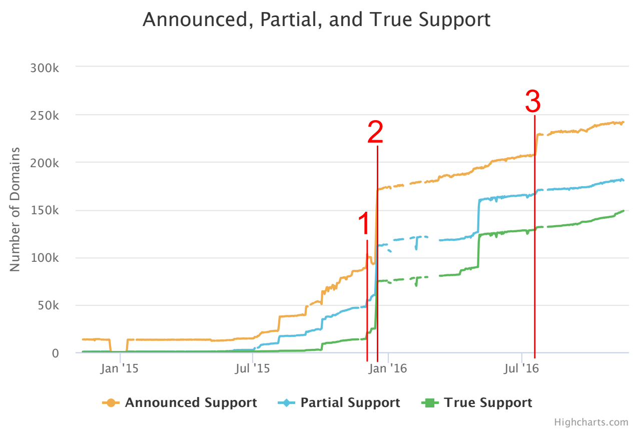

Fig. 7. Historical trend of the adoption of HTTP/2 since November 2014 from HTTP/2 Dashboard [29]

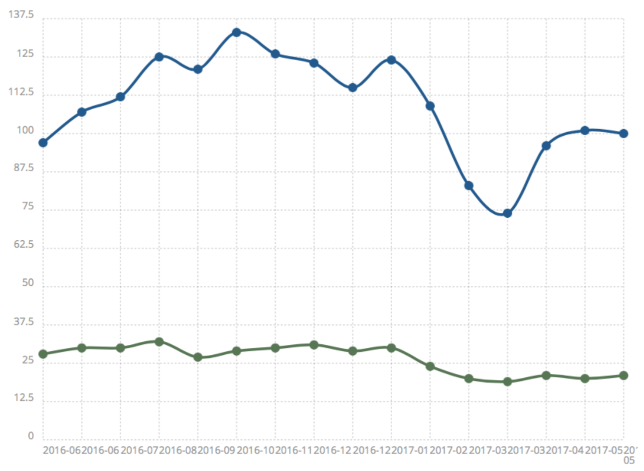

Fig. 8. Historical trend of the adoption of WebSockets using the Socket.IO API since June 2016 from BuiltWith, Blue line: Top 100.000 websites, Green line: Top 10.000 websites [19]

6 Conclusion

The introduction of HTTP/2 offered many useful changes compared to the older HTTP versions. The focus here was on better utilization of the available bandwidth, for example by preventing the HOL blocking problem and the efficient header compression. Also due to the fact that it is an official HTTP standard and that it is compatible with previous versions is contributing that the adoption will increase very quickly and more and more servers will switch to the new standard. But now to the main question of this paper, considering whether HTTP/2 really can replace WebSockets:

In our opinion this will not happen in the near future, as the deployment of both techniques pursued different objectives. WebSockets have been developed from the beginning to enable efficient bidirectional communication on the Internet without having to rely on the original request-response system. The headers are reduced to the absolut minimum and the protocol allows a wide range of applications which would have been difficult to implement through pure HTTP communication.

HTTP/2, by contrast, was developed for a general more efficient client-server communication, which does not necessarily has fully equal sides. Header compression and multiplexing allows much better network- and resource utilization than it was previously possible under HTTP/1.1. This brings tremendous benefits to server operators, as reflected by the rapid adoption of HTTP/2, although it is a relatively new standard. The approach by server-push to allow the server to send resources to the client without the explicit request of the client, but however this was in our opinion rather developed to reduce the number of necessary requests (and thus round trip times) for a faster page load time, than to stand in competition to WebSockets.

Somehow surprisingly is the very low current adoption of WebSockets. Even large companies like Facebook do not rely on WebSockets at all, although it would be a quite useful technique e.g. for the chat feature. There is an interesting comment from an Facebook web engineer who says the main reason for that is that WebSockets are not widely adopted yet and long polling is working well for their use cases [22]. It turns out that the strict necessity to use WebSockets often does not exist and it is therefore easier to save the effort for the implementation of WebSockets. Also the maximum possible compatibility often speaks for using HTTP/2.

This does not mean, however, that WebSockets are completely replaceable. Applications that require very low latency and much unforeseen communication from the server to the client are well advised to think about to use WebSockets. Section 4 showed that the communication overhead is extremely low compared to HTTP, and thus has many advantages for the server operator as well for the customer through a smoother user experience. It remains to await whether new developments in the future tend to be based on WebSockets, or whether long-polling is sufficient for most applications.

Also the further development of the HTTP standard could have a great influence on the use of WebSockets. If the standard in the next versions focuses on real equality between client and server and makes the communication even more efficient, this would really can cause making WebSockets obsolete. However, since the change from HTTP/1.1 to HTTP/2 has taken 16 years and HTTP/2 was introduced recently, this may take a very long time.

In conclusion, the future of WebSockets is not yet decided, but it remains to be seen how WebSockets will continue to spread, or if they will disappear in insignificance.

References

1. Internet Engineering Task Force (IETF). (2015). RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2). Retrieved 11:35, May 18, 2017, from `https://tools.ietf.org/html/rfc7540`
2. Head-of-line blocking. (2017, May 7). In Wikipedia, The Free Encyclopedia. Retrieved 12:37, May 22, 2017, from `https://en.wikipedia.org/wiki/Head-of-line_blocking`
3. Internet Engineering Task Force (IETF). (2015). RFC 7541 - HPACK: Header Compression for HTTP/2. Retrieved 19:00, May 18, 2017, from `https://tools.ietf.org/html/rfc7541`
4. Tim Berners-Lee. (1991). The Original HTTP as defined in 1991. Retrieved 14:15, May 22, 2017, from `https://www.w3.org/Protocols/HTTP/AsImplemented.html`
5. Internet Engineering Task Force (IETF). (1996). RFC 1945 - Hypertext Transfer Protocol – HTTP/1.0. Retrieved 14:30, May 22, 2017, from `https://tools.ietf.org/search/rfc1945`
6. Internet Engineering Task Force (IETF). (2014). RFC 7230 - Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing - Section 6.3.2. Retrieved 15:30, May 23, 2017, from `https://tools.ietf.org/html/rfc7230#section-6.3.2`
7. IETF HTTP Working Group. (n.d.). HTTP/2 Frequently Asked Questions. Retrieved 11:30, May 24, 2017, from `https://http2.github.io/faq/#why-is-http2-binary`
8. Internet Engineering Task Force (IETF). (1999). RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1. Retrieved 10:00, May 23, 2017, from `https://tools.ietf.org/html/rfc2616#section-4.4`
9. Internet Engineering Task Force (IETF). (2015). RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2) - Section 5.3. Retrieved 17:10, May 23, 2017, from `https://tools.ietf.org/html/rfc7540#section-5.3`
10. Internet Engineering Task Force (IETF). (2011). RFC 6455 - The WebSocket Protocol. Retrieved 17:30, May 23, 2017, from `https://tools.ietf.org/html/rfc6455`
11. Internet Engineering Task Force (IETF). (2011). RFC 6455 - The WebSocket Protocol - Section 1.7. Retrieved 17:40, May 23, 2017, from `https://tools.ietf.org/html/rfc6455#section-1.7`
12. W3Techs. (n.d.). Technologies Overview. Retrieved 18:00, May 23, 2017, from `https://w3techs.com/technologies`
13. Cloudflare. (2015, December 3). HTTP/2 is here!. Retrieved 18:30, May 23, 2017, from `https://blog.cloudflare.com/introducing-http2/`
14. Wikipedia. (2016, May 4). Tweet from @Wikipedia. Retrieved 18:35, May 23, 2017, from `https://twitter.com/Wikipedia/status/727910305112563713`
15. Google Security Blog. (2016, May 3). Bringing HTTPS to all blogspot domain blogs. Retrieved 18:40, May 23, 2017, from `https://security.googleblog.com/2016/05/bringing-https-to-all-blogspot-domain.html`
16. Internet Engineering Task Force (IETF). (2011). RFC 6455 - The WebSocket Protocol - Section 1.2. Retrieved 19:00, May 23, 2017, from `https://tools.ietf.org/html/rfc6455#section-1.2`
17. Internet Engineering Task Force (IETF). (2011). RFC 6455 - The WebSocket Protocol - Section 1.3. Retrieved 19:10, May 23, 2017, from `https://tools.ietf.org/html/rfc6455#section-1.3`
18. Socket.IO. (2017, April 17). In Wikipedia, The Free Encyclopedia. Retrieved 19:30, May 23, 2017, from `https://en.wikipedia.org/wiki/Socket.IO`

19. BuiltWith. (2017, May 23). Socket.IO Usage Statistics. Retrieved 19:45, May 23, 2017, from `https://trends.builtwith.com/javascript/Socket.IO`
20. The Chromium Projects. (n.d.). SPDY: An experimental protocol for a faster web. Retrieved 10:30, May 24, 2017, from `http://dev.chromium.org/spdy/spdy-whitepaper`
21. Peter Lubbers, Frank Greco. (n.d.). HTML5 WebSocket: A Quantum Leap in Scalability for the Web. Retrieved 12:30, May 24, 2017, from `https://websocket.org/quantum.html`
22. Ronen Pinko. (2014, September 20). Quora board entry. Retrieved 11:30, May 27, 2017, from `https://www.quora.com/Why-does-Facebook-use-long-polling-instead-of-WebSocket-in-order-to-instant-chat`
23. HTTP/2. (2017, May 27). In Wikipedia, The Free Encyclopedia. Retrieved 12:19, May 27, 2017, from `https://en.wikipedia.org/wiki/HTTP/2`
24. Brian Jackson. (2016, April 25). KeyCDN Enables HTTP/2 HPACK Compression - Huffman Encoding. Retrieved 14:00, May 25, 2017 `https://www.keycdn.com/blog/http2-hpack-compression/`
25. Wikimedia Commons. HTTP persistent connection, client-server connection schema. (c) CC0. Retrieved June 1, 2017 from `https://commons.wikimedia.org/wiki/File:HTTP_persistent_connection.svg`
26. Wikimedia Commons. HTTP pipelining, client-server connection schema. (c) CC0. Retrieved June 1, 2017 from `https://commons.wikimedia.org/wiki/File:HTTP_pipelining2.svg`
27. Google. Screenshot of HTTP/2 101 (Chrome Dev Summit 2015). (c) CC-BY. Retrieved June 1, 2017 from `https://www.youtube.com/watch?v=r5oT_2ndjms&t=275s`
28. W3Techs. Usage of HTTP/2 for websites. Retrieved May 23, 2017 from `https://w3techs.com/technologies/details/ce-http2/all/all`
29. HTTP/2 Dashboard. HTTP/2 Adoption. Retrieved May 23, 2017 from `http://isthewebhttp2yet.com/measurements/adoption.html`