Impacts of the HTTP/2 protocol for large scale web environments

Martin Leucht, James Gratchoff
Master SNE, UvA

March 23, 2015

1 Introduction

HTTP/1.1 is present since 1999 and it is such a big protocol and has been reviewed so many time that the IETF has split the original RFC (2616) into six different ones (7230-7235) describing the whole protocol in details. After more than 15 years of use it was time for a change. All the tricks not to slow the HTTP/1.1 protocol can be forgotten as on February 18th 2015, the specification for the new HTTP protocol HTTP/2 (and HPACK), has been formally approved by the IESG and is on the way to become an RFC standard. The main focus during the development period of the successor of HTTP/1.1 was to improve the performance, and thus provide a better user experience. The performance improvement is mainly based on how the packets are sent over the wire within a HTTP/2 session. HTTP/2 data is sent in binary format and is based on a multiplexing mechanism that allows a single connection for parallelism. Concerning security, HTTP/2 will not make the use of TLS mandatory. However, leading browsers firms, Mozilla Firefox and Google Chrome have already mentioned that HTTP/2 will only be implemented over TLS.
There are already web client and server implementations that support the final HTTP/2 specification. This research is intending to show what are the impacts of implementing HTTP/2 with TLS compared to HTTPS in a large environment.

2 Research Questions

To conduct the research the following research questions have been formulated:

How do the new features of the HTTP/2 protocol improve the performance for frequently visited webpages/webserver?

That question can be narrowed down to some sub questions:

- What are possible drawbacks that can occur for large web service providers when switching from HTTP/1.1 to HTTP/2 ?
- What is the predictable impact that could be related to changes in the infrastructure of Web service providers?
- What is the difference in bandwidth utilization between HTTP/1.1 and HTTP/2 considering the size ratio of header and data for multiple concurrent sessions on a server?
- How much can HTTP/2 decrease the amount of data when considering header compression and using multiple streams within one TCP Session compared to HTTP/1.1 on a server ?
- What is the impact regarding the response time experienced by a client when the number of concurrent requests / clients increases seen from three different geographical locations?
- What is the impact to the load and CPU utilization to the server when conducting the benchmark tests?

3 Scope

The scope of this research is to conduct benchmarks of the HTTP/1.1 and HTTP/2 protocol, using different static and dynamic parameters and compare them among each other. Static parameters

are the distance between client/server (RTT) and the page size, whereas dynamic parameters are the number of requests and the number of concurrent clients. Therefore we will implement three different HTML pages with different sizes and benchmark them from three different geographical locations. The test will be implemented using the secured version of the protocols with TLS enabled. We do not focus on new features of the HTTP/2 protocol like server push capabilities and flow prioritization.

4 Literature review

4.1 HTTP/1.1 drawbacks

Specified in multiple RFCs(7230-7235), HTTP/1.1 is a standard protocol for web-browsing that is now used for more than 15 years. This protocol has been reviewed a number of times and contains a high number of options. It uses a client-server model where the clients is most of the time a browser that is getting data from a server located in a different location. HTTP is the essence of the web and most of the users are using this protocol on a daily basis. Since the early beginning of the protocol, users complained about the time that takes a page to load. One big reason of this problem was not due to the protocol but to the speed and the reliability of the Internet. However since the web has changed and more and more users have a high and reliable connection to the Internet. This and the growth of clients on the web has led to the increase of web pages size and the number of elements on a web page. HTTP was not designed for this use as indeed it is inadequately using the TCP protocol by retrieving one element from a web page using one TCP connection. It is thus not taking full advantage of the high performance TCP protocol and thus it is leading to a slower loading time. The loading time is also highly dependent on the Round Trip Time (RTT) and some efforts have been done on improving this aspects over the years by adding geographical redundancy however the problem is still present for low cost organisation that are only able to develop server in a single location. //SPEAK ABOUT TCP head of line blocking here// This latency directly affect the clients and can be critical for your website (e.g. users leaving the page as it takes too long to load). So over the years, web developers have tried to reduce this critical factor that is latency. In order to do so the developers have been tried to adapt themselves to the protocol by trying to reduce the latency by doing many tricks or workarounds. The most famous of these workarounds are:

- Spriting: The fact to create an image containing many pictures destined to be present on the website and let the browsers display (via CSS or Javascript) the picture wanted by cropping/-cutting out the single image from the big one.
- Sharding: In order to overcome the problem of increasing TCP connections per domain, developers started to create several domains, holding different part of the website. This decreases the page loading time by reducing the number of connection per host. Thus leading to a better performance of the HTTP/1.1 protocol.
- Concatenation: In order to reduce the number of TCP connections, devellopers started to concatenate files (e.g. javascript) into one big files.
- Inlining: By embedding the data straight in the CSS in base64 format it avoids to send picture and thus creating new TCP connection.

All this workarounds were needed as the page loading time and the number of requests were increasing so much that the web was slowing down even though more and more people had access to a high and reliable connection. After 15 years, it was time for a change. That is why the IETF created a working group named HTTPbis that started working on a new protocol. Beforehand Google started working on a new protocol, called SPDY, that was aiming to encounters the problems from the HTTP/1.1 protocol. The HTTPbis group started working from a working concept of protocol in the name of SPDY/3 (draft). And this was the start of HTTP/2.

4.2 HTTP/2 improvements

This section is subdivided into several ones describing how it improves from its predecessor HTTP/1.1 and emphasis on the reasons to move to the new HTTP/2 protocol for large scale environments.

4.2.1 Binary format

HTTP/1.1 is based on a text/ascii format which is an advantage for humans to read and thus to debug the protocol. It is described by Raymond as "easy for human beings to read, write, and edit without specialized tools"(http://www.catb.org/esr/writings/taoup/html/ch05s01.html). However for computers such as clients and server, ascii is not their mother tongue. Indeed, without any surprises, computers are using binary as a format for exchange. HTTP/2 is using this format. Binary is know to be much more efficient for binary structures. It is indeed hard to define the start and the end of a field in text based protocols. However with binary format it is much more natural. Binary will then improve the structure of the protocol and thus the efficiency of the protocol "HTTP/2 also enables more efficient processing of messages through use of binary message framing."(https://datatracker.ietf.org/doc/draft-ietf-httpbis-http2/?include_text=1). In order to overcome the difficulty of debugging the binary protocol, tools such as curl have added support for HTTP/2 and Wireshark have created extensions to decode the network stream.

4.2.2 Multiplexing and priorities

The use of stream is a major enhancement of the HTTP/2 protocol. A stream is described in the specification as "an independent, bi-directional sequence of frames exchanged between the client and server within an HTTP/2 connection." The stream identifier, present in the header format under as an integer format, will associate each frame belonging to the same stream. One HTTP/2 connection can contain several concurrently open stream, that can each be closed by the client or the server. Streams are multiplexed which can mean that they do not arrive at the same order as they have been sent. It is the role of the client to put this streams back together in a correct order to process the data. Each stream has a priority that can be set by the client in the HEADERS frame that opens the stream. This priority can be changed to re-prioritize a specific stream. This can permit to avoid the head of the line blocking and allow an endpoint to express how it would prefer to retrieve data when managing multiple concurrent streams. A stream can also be dependent on other streams by setting the stream dependency parameter.

4.2.3 Header compression

One of the problem of HTTP/1.1 described earlier is that more and more elements are fetched per web page. If too elements are close in type and location the headers will be very similar and

"thus the redundant header fields in these requests unnecessarily consume bandwidth, measurably increasing latency. " (http://http2.github.io/http2-spec/compression.html) That is why header compression is a good solution to this problem. However HTTPS and the SPDY compression mechanism have been vulnerable to the BREACH and CRIME attacks. That is why the HTTPbis group has created HPACK (Header Compression, that is the specification that comes with HTTP/2. HPACK is described as "simple and inflexible." It eliminates redundant header fields and prevent security issues.

4.2.4 Flow control

The flow control is implemented in HTTP/2 by assigning an integer value to a window that will define how many octets of data the sender (server) is able to transmit. In that way the server is taking in consideration the buffering capacity of the receiver. This has enormous advantages as receivers on the web can be of different nature (e.g.Smartphones, laptop) with different connection. Flow control control can either operate on the entire connection or on each individual stream. The default value is 65535 octets. In order to reassign this value the receiver (client) can send a SETTINGS frame with his flow control integer value.

4.2.5 Server push

HTTP/2 allows the server to send extra information with a requested information required by a client. This functionality permits to increased the overall loading time by anticipating what the client will need before it asks for it. In that way the client will already possess the information when asking for it. This is called the server push mechanism. This mechanism is not required but can improve the user experience. The client must allow the server to do so. This allow the client to stay in control and indeed the client is able to terminate that pushed stream by sending a RST_STREAM.

4.2.6 Security

4.3 Related work

The new HTTP2 protocol has been based on the SPDY protocol developed mainly by Google. As shown by Google [1], the SPDY protocol meets its expectations by reducing the loading time of web pages by 55%. Other people have tried to look into the protocol and one of the most interesting analysis has been done by Servy[2]. Servy evaluated the performance of the web servers implementing the SPDY protocol comparing it to HTTP/1.1 and HTTPS. The load testing tool used for this benchmark was the NeoLoad 4.1.2. His results showed that the implementation of SPDY increases by a factor of 6 the number concurrent of users possible before errors start showing up in comparison to HTTP and HTTPS. A contradictory study showing some boundaries of implementing SPDY has been done by Podjarny[3]. He shows that most of the websites use different domains and as SPDY works on a per-domain basis it does not necessarily help it to be faster. Finally, Wang et al.[4] have investigated the performance of SPDY for the improvements of the protocol compared to HTTP/1.1. This study highlights that SPDY is much faster since its benefits from the single TCP connection mechanism. However, they also mention that SPDY degrades under high packet loss compared to HTTP. Concerning the new standard HTTP/2 a few benchmarks have been performed by the creators of different client/server platforms. They reach the same conclusion for SPDY.

However, studies on comparisons between HTTP/2 (draft-ietf-httpbis-http2-14 [8]) and HTTP/1.1 with regards to concurrent clients and increasing amount of requests in different geographical locations and different page sizes or amount of elements a webpage contains, have not been conducted yet.

5 Requirements

In order to conduct this research we use one server that runs a webserver and responds to both versions of HTTP, HTTP/2 and HTTP/1.1 [7]. That instance will run on of our student servers. AWS medium-instances, provided by the Amazon Elastic Compute Cloud (EC2) [5] infrastructure, will be used as clients to perform several benchmark tests that try to cover real life scenarios.

6 Method

For our research we will setup three AWS Amazon micro instances (Europe/Asia/North America) that will act as HTTP clients (HTTP/1.1 and HTTP/2). The server itself, that will serve HTTP/1.1 and HTTP/2 requests is located in the OS3 lab and is one of our student server. The server will provide three different web pages that have different sizes (small/medium/large). As a server software for testing HTTP/2 (h2c-14 [8]) requests we will use nghttp [9]. For HTTP/1.1 requests we will use Apache2 [10]. In order to benchmark HTTP/2 we will use h2load [11] and for HTTP/1.1 Apache Benchmark (ab [12]). Despite the will not to use different benchmarks tools to minimize the repercussions on the results, we decided to use these tools because both provide similar output (data/header size ratio, requests per second, mean time for each request, etc) and are capable to simulate concurrent sessions.

7 Implementation

Three HTML of different sizes have been created to be compared. These pages reflect the sizes that can be encountered in most common websites and has been derived from top 100 websites statistics [13]. The pages size are:

title	size (bits)	number of requests
small.html	20458	3
medium.html	624048	8
large.html		49

Other fixed parameters have been set-up: $explanationoftheotherparameters$ Dynamic parameters $Whataretheparameterswewillbeplayingwith$

In order to have relevant results, the tests have been performed ten times and the average results will be presented in the next section. In order to make the benchmarking easier and more coherent some scripts have been implemented. $explanationofthescriptswrittenbyMartinhere$

8 Results analysis

9 Conclusions

10 Further Work

References

[1] A 2x Faster Web. (2009). [online] Chromium Blog. Available at: http://blog.chromium.org/2009/11/2x-faster-web.html [Accessed 20 Feb. 2015].

[2] Servy, H. (2015). Evaluating the Performance of SPDY-enabled Web Servers. [online] Neotys.com. Available at: http://www.neotys.com/blog/performance-of-spdy-enabled-web-servers/ [Accessed 20 Feb. 2015].

[3] Podiatry, G. (2015). Guy's Pod Blog Archive Not as SPDY as You Thought. [online] Guypo.com. Available at: http://www.guypo.com/not-as-spdy-as-you-thought/ [Accessed 20 Feb. 2015].

[4] Wang et al. (2014). How Speedy is SPDY?, 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). [online] usenix.org. Available at: https://www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-wang_xiao_sophia.pdf [Accessed 20 Feb. 2015].

[5] Amazon Elastic Compute Cloud (Amazon EC2) (2015). Available at: http://aws.amazon.com/ec2/ [Accessed 20 Feb. 2015].

[6] Firefox Beta Notes (2015). Available at: https://www.mozilla.org/en-US/firefox/36.0beta/releasenotes/ [Accessed 20 Feb. 2015].

[7] HTTP/2 client/server implemenations (2015). Available at: https://github.com/http2/http2-spec/wiki/Implementations [Accessed 20 Feb. 2015].

[8] Hypertext Transfer Protocol version 2 draft-ietf-httpbis-http2-14 (2014). Available at: https://tools.ietf.org/html/draft-ietf-httpbis-http2-14 [Accessed 4. March 2015]

[9] HTTP/2 experimental server (2015). Available at: https://nghttp2.org/documentation/nghttpd.1.html [Accessed 4. March 2015]

[10] The Apache HTTP Server Project (2015). Available at: http://httpd.apache.org/ [Accessed 4. March 2015]

[11] Benchmarking tool for HTTP/2 and SPDY server (2015). Available at: https://nghttp2.org/documentation/h2load.1.html [Accessed 4. March 2015]

[12] Apache HTTP server benchmarking tool (2015). Available at http://httpd.apache.org/docs/2.2/programs/ab.html [Accessed 4. March 2015]

[13] Httparchive.org, (2015). HTTP Archive - Trends. [online] Available at: http://httparchive.org/trends.php [Accessed 11 Mar. 2015].