HTTP/1.1 pipelining vs HTTP2 in-the-clear: performance comparison

Romuald Corbel
Orange Labs
Lannion, France
romuald.corbel@orange.com

Emile Stephan
Orange Labs
Lannion, France
emile.stephan@orange.com

Nathalie Omnes
Orange Labs
Lannion, France
nathalie.omnes@orange.com

ABSTRACT

The average Web Page size is constantly increasing: it doubled between January 2012 and January 2015 [1]. In such a context the transport of the Internet traffic becomes very challenging. To moderate the impact of the increase of traffic on end-users' Quality of Experience (QoE), the IETF specified the protocol HTTP2, which optimizes the transfer of the HTTP1 protocol.

In this paper we provide a comparison of HTTP1 and HTTP2, both in terms of functionalities and in terms of Page Download Time (**PDT**). As end-to-end encryption is not always required, this comparison is provided in-the-clear. To avoid any bias in the comparison, our measurements are made with the same hardware, the same software and the same transport conditions (multiplexing over a single TCP connection per domain).

Based on our measurement, we show that HTTP2 always highlights better performances that HTTP1. More precisely, we observe the HTTP2 PDT remains 15% lower than HTTP1 PDT as the network delay increases. Furthermore, we observe that the ratio of HTTP2 PDT to HTTP1 PDT decreases as the packet loss increases, showing that HTTP2 is more resilient to packet loss than HTTP1.

Keywords

HTTP1, HTTP2, H2C, Apache, Nghttp2, page download time, average web page, performance

1. INTRODUCTION

The average Web Page size has doubled over the last 3 years [1]. The main part of the increase of traffic is naturally driven by the increase of the resolution of the video and the complexity of e-commerce applications. In addition, there is a hidden part which comes from optimizations made by web sites [2]. As an example, domain sharding [3] (content storage in multiple sub-domains to increase the number of concurrent connections) increases the throughput available between a browser and a Web application but adds a lot of connection overhead as it duplicates headers, cookies, CSS files and JavaScript files exchanges.

HTTP1 has been designed in 1990 to request HTML pages from a text-based web browser. Modern Web applications have stronger requirements in term of interactivity and must adapt to extremely varying network conditions in mobile situations. However, while downloading the web page requires several requests, these requests cannot be truly parallelized. Even with HTTP/1.1 pipelining the response to a request *n* has to wait until the response to the request *n-1* is sent. This is the well-known HTTP head of line blocking issue.

To overcome this issue, Google experienced a new protocol named SPDY [4]. The initiative was reused by the IETF to specify HTTP2. HTTP2 [5, 6] does not change HTTP semantics. HTTP2 transports HTTP headers and data in separate frames. HTTP2 headers compression [6] optimizes headers carrying on-the-wire, solving HTTP headers repetitions issue. Both HTTP/1.1-pipelining and HTTP2 support multiplexing the requests in the same TCP connection. In HTTP2, requests and responses are further numbered. This allows interleaving the responses in parallel and solves the HTTP head of line blocking issue.

A number of papers have presented the benefits of SPDY or HTTP2 [7, 8]. Since existing HTTP2 stacks are implemented over TLS, most of the time these studies compared HTTP2/TLS with HTTP1/TLS [9]. In this paper we chose to compare the performance of the protocols HTTP1 and HTTP2 in-the-clear over TCP. We use the new Apache module [10] mod_http2 and the Nghttp2 [11] for this purpose.

Web sites are progressively moving to HTTPS to protect end-users privacy. In the meantime, they are migrating to HTTP2 to improve both network performance and end-user experience. Although the Internet is moving to encryption, there are relevant use cases where TLS encryption is not required while HTTP2 is still desirable:

- Servers located behind a reverse proxy terminating encrypted HTTP2 connections and receiving the

HTTP2 traffic in-the-clear. Currently the reverse proxy must translate HTTP2 into HTTP1.

- Server-to-server communication, like datacenter interconnection, where the HTTP2 traffic is encrypted using other techniques than TLS like IPSEC.
- Migration to encryption and to HTTP2 simultaneously. This is something complex [12]. So it is desirable to separate the deployment of TLS from the migration to HTTP2.

The paper is organized as follows: section II introduces the protocol HTTP2 and HTTP1 pipelining. The methodology and the measurement platform are presented in section III, while measurements are presented and analyzed in section IV. We conclude the paper and introduce future work in section V.

2. HTTP2 PROTOCOL

HTTP2 was developed by the Hypertext Transfer Protocol working group of the Internet Engineering Task Force (IETF). The specification started end of 2012 based on Google experimental SPDY protocol [4] with main objective to improve HTTP1 performance. HTTP2 is thus an optimization of the transfer of HTTP1 messages already deployed by the main stakeholders of the Web. HTTP2 was approved and published by the IETF as RFC7540 [5] and RFC7541 [6] in May 2015.

HTTP2 is defined both for HTTP URIs (i.e. without encryption) and HTTPS URIs (over TLS). Most client implementations, such as Firefox or Chrome, only support the transport of HTTP2 over the TLS protocol.

The main characteristics of HTTP2 are the following:

Connection: A HTTP2 connection multiplexes HTTP1 exchanges between a client and a server over the same TCP or TLS connection.

Stream: Each HTTP1 request and its responses are carried in a uniquely identified HTTP2 stream.

Frame: HTTP2 carries HTTP1 headers and body in separate frames. To achieve this, each HTTP1 message is split into one binary control and several DATA frames.

Control: HTTP2 specifies 9 different control frames which have the priority over DATA frames (HEADERS, PRIORITY, RST_STREAM, SETTINGS, PUSH_PROMISE, PING, GOAWAY, WINDOW_UPDATE and CONTINUATION). The scheduling depends on the type of application.

Headers: HTTP1 Header fields are compressed based on indexation and delta serialization technics specified in [6].

Interleaving: Thanks to the unique stream identifiers and to the framing, HTTP1 requests can be processed in parallel. Frames interleaving is further allowed within each stream. This resolves the HTTP1 pipelining limits.

Streams and frames thus need to be scheduled.

Flow Control: Control frames are always sent first. When the flow control is activated, it prioritizes the sending of DATA frames according to their level of priority and the window size of each stream.

Push: HTTP2 is not only an optimization of HTTP1. It introduces the PUSH_PROMISE mechanism which allows the server to send contents in advance in the cache of the browser.

Transport: The HTTP2 specification [5] requires the transport of HTTP traffic over a single TCP or a single TLS connection.

Migration from HTTP/1.1 to HTTP2

The HTTP2 connection is established between a client and a server. Because the client and server implementations may differ, HTTP2 offers 3 modes of coexistence and migration with HTTP1:

- Upgrade of HTTP1 to HTTP2 using the HTTP1 “upgrade” header. It requires the negotiation of the upper layer protocol using the “H2C” value. A server which does not support HTTP2 ignores it;
- Direct connection in the clear: Direct mode supposes a prior knowledge of the support of HTTP2. It does not require any negotiation;
- Transport over TLS: requires the negotiation of the HTTP version using ALPN [13] TLS extension.

3. METHODOLOGY

Both HTTP2 and HTTP1 carry the semantic of HTTP. Nevertheless their transport sub-layers differ.

This section firstly studies the evolution of the HTTP exchange over the Internet and determines the characteristics of an Average Web Page (**AWP**). Then, it presents the test platform and the measurements. Finally it details the measurement tools configuration.

3.1 AWP Characteristics

The distribution of the requests types (mime-type) did not change from 1994 to 2004 as shown in [14]. According to [15], the main changes are the pages constituents which move from mostly one file to hundreds. A page is now a mashup of contents from distributed web services hosted on different domains (ads, news, images in static domains, dynamic pages, CDNs …). In 2014 the AWP of the top 300,000 pages is 1.8 MB [16]. With regard to the packets’ sizes, the minimal number of packets per flow is increasing

[17]. To capture this evolution, we retain a minimum of 40 packets number per flow.

Table 1 : AWP characteristics

mime type	file size (kb)	requests number	response size (kb)
HTML	56	9	6
CSS	63	7	9
JavaScript	329	20	16
Images	1310	56	23
Flash	90	1	90
other	152	7	22
Total	2000	100	NA

The table above summarizes the measured characteristics of HTTP archive [18] in May 2015: the download of an AWP generates 100 requests and downloads 2MB of data from 16 different domains or sub-domains.

3.2 Web Page Scenario and Scheduler

To compare HTTP2 and HTTP1 performances, we measure the time needed to download the same AWP with HTTP1 and HTTP2, under the same hardware, software and network characteristics.

Scenario: The repartition of the AWP components (contents and scripts) over the 16 domains is given by the table 2. The first line of the table gives the mime-type of a request. The first column indicates the domain name. Finally each cell indicates the number of times this request is sent to this domain. The size of the files returned in the responses to each request is given by the table 1.

Table 2 : AWP Requests scheduling

Domain\file type	html	css	js	img	flash	other
domain1.com	1	1	1			
domain2.com	1	1	1			
domain3.com			4			1
domain4.com			4			
domain5.com			4	3		
domain6.com	1	1				
domain7.com	1	1				
domain8.com	1	1	1			1
domain9.com	1	1	1			
domain10.com	1					4
domain11.com			4	3		1
domain12.com	1	1				
domain13.com	1				1	
domain14.com				10		
domain15.com				10		
domain16.com				30		

The domains 14, 15 and 16 receive half of the requests which are static images. 25% of the domains receive more than 7 requests.

Client scheduler: In both the cases of HTTP1 pipelining and of HTTP2, the client opens an unique TCP connection with each domain and sends immediately the entire HTTP requests of the scenario to this domain. To avoid any bias the client deactivates any application-level optimization of HTTP2 (HTTP2 PUSH_PROMISE and PRIORITY are disallowed).

3.3 Measurement Platform

Our measurement platform is made of a HTTP server, a HTTP client requesting a page on the server and a network interconnecting both of them.

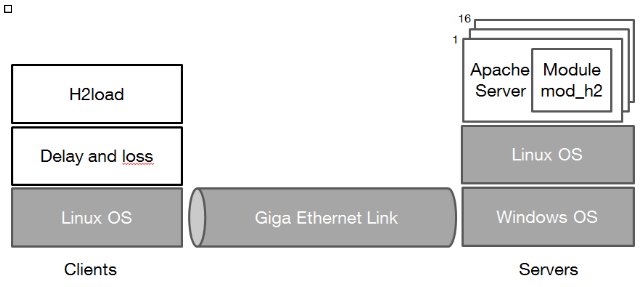

Figure 1: Platform

It is important to notice that servers are implemented as virtual machines on the same computer. By consequence all requests and responses share the same bufferization both for network card and web server CPU. This is emphasized by the double OS (Windows + Linux) crossed by the traffic. As an example, as all clients start simultaneously and all servers share the same 1 Gbps line card of the same physical machine, the 40 trains of 2 MB of data of the responses suffer from an additional average delay of 320ms.The client side suffers from the bufferization effect but is not impacted by virtualization.

The limitations of the platform noticeably increase the measured delays, however they allow the comparison of HTTP1 and HTTP2.

3.4 Measurement tools

HTTP Clients: The HTTP client must be able to send either HTTP2 or HTTP1 unencrypted traffic. We selected h2load [19] amongst other solutions because it generates both HTTP1 and HTTP2 in-the-clear and additionally because it provides the measurement parameters needed to compute the metrics (payload bytes, header bytes, time to download data…).

DNS server: The local DNS server resolves the name of the 16 domains of the Web servers (domain1.com to domain16.com). It runs on the same machine as the HTTP servers.

Network: Clients and servers are connected back-to-back on an Ethernet 1Gbps network. The TCP initial window size is always 65535 bytes, while the MTU is 1500 bytes. Network impairments (delay and packets lost) are generated with the Linux tc command. As an example, the command below introduces a delay of 50 ms and a packet loss ratio of 0.3%.

```
#tc qdisc add dev eth0 root netem delay 50ms loss 0.3%
```

HTTP Server: The Web server must support both HTTP1 and HTTP2 in-the-clear. We use an Apache HTTP server version 2.4.17 with the module mod_http2 [10].

Implementation: Clients run on a Linux Debian OS. Each domain has its own Apache server. They run in a VirtualBox virtual machine (VM) hosted in a PC having 8 Go of the RAM and a Intel Xenon processor with 8 cores.

Metrics: We measure the page download time. The page download time represents the duration of the download of all the contents. It is the time between the first request and the reception of the last file (e.g. the last .img file of the domain 16) and the sending of the .html request of the domain 1).

The table below depicts the end-to-end delay we introduce during our measurements. They are representative of mobile and fix networks.

Table 3 : Delay

delay	Use Case
10ms	f optical
25 ms	ADSL
50 ms	LTE
75 ms	ADSL + intercontinental
100ms	3G
150ms	3G + intercontinental
250ms	2G CDN local
300ms	2G + intercontinental
500ms	AMEA + intercontinental
1 second	Very long delay

The table below depicts the packet loss percentage we introduce in our measurements. These are representative values extracted from the estimation made by the tool Queen [20]:

Table 4 : Packet Loss Rate

Packet Lost	Use case
0.1%	Europe : < 0,1% in 50% of the cases
0.3%	Europe : < 0,3% in 80% of the cases
0.5%	Africa : < 0,5% in 50% of the cases
1%	Europe : < 1% in 85% of the cases
2%	Europe : < 2% in 95% of the cases
5%	Europe : < 5% in 95% of the cases

Calibration: We firstly measured that the sustainable number of clients supported individually by each HTTP server is 80. To reduce the side effects due to servers resources we limit the number of clients to a maximum of 40 during the measurements.

Measurement: Measurements are scheduled and monitored by a nodeJS script. One measurement consists in 40 clients in parallel downloading the HTTP page of 2MB over 16 HTTP servers. Hence a measurement generates 640 TCP connections in parallel. The script aggregates H2load output parameters.

Filtering: Each measure is run 20 times in sequence. We discard any results having HTTP errors or abnormal PDT value (when |pdt – mean(pdt)| > 2 sd(pdt)).

4. Results Analysis

In this section, we present the results of our measurements.

More precisely, we compare the Page Download Time (PDT) with HTTP1 and HTTP2 respectively as the packet loss and network latency vary.

At first glance, applying the bandwidth delay product [21], the default TCP window size is responsible for increasing the PDT when the network delay is larger than 250ms. However, during the measurement, as the responses suffer from an additional average delay of 320ms, we estimate the TCP window size starts increasing the PDT when the network delay is over 75ms.

4.1 PDT variation with network delay

The following figure depicts the variation of the average HTTP1 PDT (resp. HTTP2 PDT) as the network delay increases. The average is computed over all measures for all packet loss values.

This figure shows that both HTTP1 and HTTP2 PDT increase as the network delay increases which is just common sense. Nevertheless, we show that HTTP2 always performs better than HTTP1 excepted for one value.

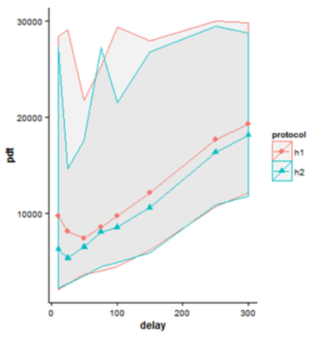

Figure 2: average HTTP1 & HTTP2 PDT for all delays

The figure below presents the PDT ratio of HTTP1 and HTTP2 for packet loss of 1 %. When the network delay is 10ms, HTTP2 is up to 2 times faster than HTTP/1.1. Then it decreases abruptly. PDTs are equal for a network delay of 50ms. Later, as the network delay increases, the ratio is further rather stable and HTTP2 is 15 % faster than HTTP/1.1.

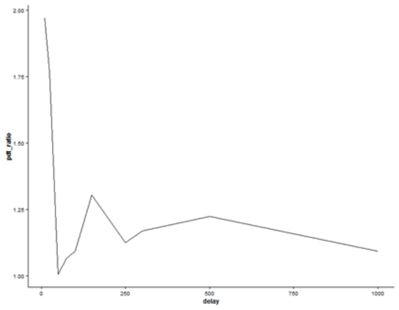

Figure 3: PDT ratio for 1% packet loss

4.2 PDT variation with Packet Loss

Let us now concentrate one the variation of HTTP1 PDT (resp. HTTP2 PDT) with the packet loss. In the following figure, we plot this variation for different network delay values: 25ms, 100ms and 250ms.

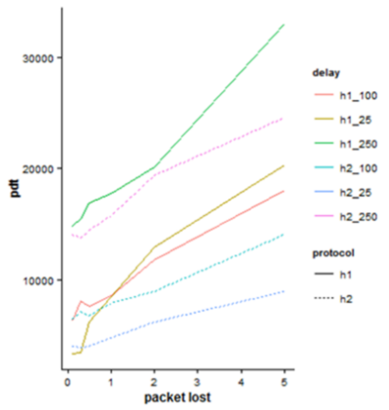

Figure 4: HTTP1 & HTTP2 PDT vs packet loss

This figure shows that both HTTP1 and HTTP2 PDT increase as the packet loss increases which is once again common sense. Furthermore, we observe that the ratio of HTTP2 PDT to HTTP1 PDT decreases as the packet loss increases.

To confirm this result, we compute the average PDT of HTTP1 (resp. HTTP2) for a fixed packet loss over all measures for all network delay values. The result is presented in the figure below. It shows that **HTTP2 is more resilient to packet loss than HTTP1.**

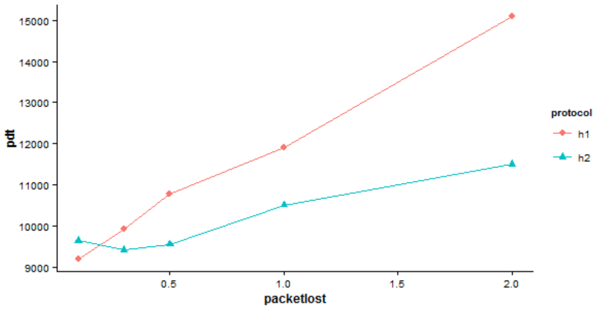

Figure 5: average HTTP1 & HTTP2 PDT for all delays

The following figure presents the number of measurements for which the PDT lies in the interval [n ; n+1[seconds and for which the packet loss is superior or equal to 1%. This figure clearly shows that HTTP2 performs better in case of lossy network, as the number of measurements for low values of the PDT is significantly better for HTTP2 as compared to HTTP1.

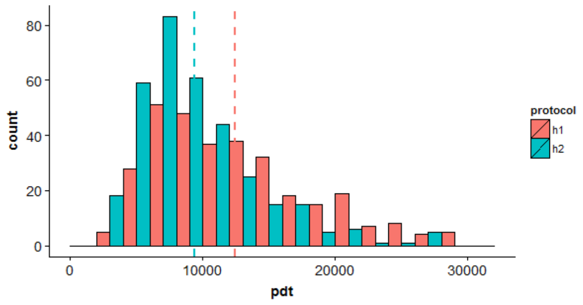

Figure 6: H1&H2 values with packet loss ≥1%

4.3 PDT variation with both packet loss and network delay

The figure bellow represents 3D curves showing HTTP1 (resp. HTTP2) PDT variation with network delay and packet loss.

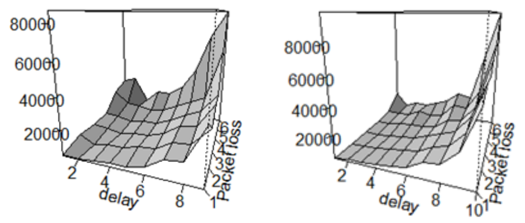

Figure 7: H1 & H2 mean PDT (loss, delay)

These figures highlight that the HTTP2 behavior is more stable than HTTP1 as network delay and packet loss increases.

5. CONCLUSION

In this article, we have firstly compared HTTP1 and HTTP2 in terms of functionalities, in particular with regard to the HTTP1 head of line blocking issue.

We have secondly studied the evolution of web pages characteristics and determined the constituents of an average web page. We have additionally chosen packet loss and network delays that are representative to real conditions.

Based on these parameters we have compared HTTP1 and HTTP2 performances under the same hardware, software and network configurations. We more precisely depict the page download time of an average web page with HTTP1 pipelining and HTTP2 in-the-clear as network delay and packet loss vary.

Thanks to our measurements, we show HTTP2 always performs better than HTTP1, with shorter page download times. The HTTP2 page download time increases more slowly than HTTP1 as the network delay increases, remaining 15% lower as compared to the HTTP1 PDT. The HTTP2 page download time also increases more slowly than HTTP1 as the packet loss increases. In this last case, we show that the two protocols have different behaviours highlighting a much better resiliency of HTTP2 towards packet loss.

Our results thus highlight the benefits of moving Web servers from HTTP/1.1 to HTTP2. This benefit increases substantially when it is coupled with the migration of a reverse proxy to HTTP2 over TLS. Server-to-server exchanges relying on encrypted communications would further benefit from the better performance of HTTP2 in-the-clear.

Yet our results are only derived from measurements between clients and servers emulated on two machines. They shall be confirmed with further measurements and with higher TCP windows size values in the clients. Further tests are also required to characterize the HTTP2 performance in mobile situation, the gain of performance provided by HTTP2 push and priority mechanisms and finally the robustness of transporting the HTTP2 traffic on a unique long duration TCP connection.

6. REFERENCES

[1] The overweight web: Average web page size is up 15% in 2014

[2] NGINX, HTTP/2 for Web Application Developers, September 16th 2015

[3] GTmetrics, PageSpeed: Parallelize downloads across hostnames

[4] M. Belshe & R. Peon, SPDY Protocol, draft-mbelshe-httpbis-spdy-00, Feb 2012

[5] M. Belshe, R. Peon & M. Thomson, Hypertext Transfer Protocol Version 2 (HTTP/2), RFC 7540, May 2015

[6] R. Peon & H. Ruellan, HPACK: Header Compression for HTTP2, RFC 7541, May 2015

[7] J. Padhye & H. F. Nielsen, A comparison of SPDY and HTTP performance, Jul 26, 2012.

[8] Jeffrey Erman & al., Towards a SPDY'ier Mobile Web, IEEE/ACM Transactions on Networking, Vol. 23, No. 6, Dec 2015

[9] HTTP2 Implementations

[10] HTTP/2 in Apache httpd

[11] Nghttp2: HTTP/2 C Library, Feb 16th, 2015

[12] Valentin Bartenev, NGINX, 7 Tips for Faster HTTP/2 Performance, October 26, 2015

[13] S. Friedl, A. Popov, A. Langley & E. Stephan, TLS Application-Layer Protocol Negotiation Extension, RFC 7301, July 2014

[14] A. Williams, M. Arlitt, C. Williamson & K. Bark, Web Workload characterization : ten years later, 2003

[15] Joachim Charzinski, Traffic Properties, Client Side Cachability and CDN Usage of Popular Web Sites, Proc. MMB/DFT 2010, Essen, Germany, Mar. 2010, pp. 136-150

[16] websiteoptimization.com, Average Web Page Breaks 1600K, Jul 2014.

[17] P. Owezarski and N. Larrieu,Internet traffic characterization – an analysis of traffic oscillations, 2004

[18] HTTP archive, May 2015 Alexa STAT

[19] Nghttp2, h2load

[20] Y. Angela Wang, C. Huang , J. Li & K. W. Ross, Queen: Estimating Packet Loss Rate, Microsoft technical report

[21] High Performance Browser Networking, Building Blocks of TCP

[22] D. Stenberg, Mozilla, TCP Tuning for HTTP, draft-stenberg-httpbis-tcp-01, Dec 21, 2015

[23] HttpWatch, A Simple Performance Comparison of HTTPS, SPDY and HTTP/2, Jan 16, 2015