The Curious Case of Parallel Connections in HTTP/2

Jawad Manzoor
Université Catholique de Louvain
Belgium
Email: jawad.manzoor@uclouvain.be

Idilio Drago
Politecnico di Torino
Italy
Email: idilio.drago@polito.it

Ramin Sadre
Université Catholique de Louvain
Belgium
Email: ramin.sadre@uclouvain.be

***Abstract*—Web pages and web-based services are becoming more and more complex. The average page size for the Alexa top 1000 websites in 2016 has reached 2.1 MB and fetching a page requires requests for 128 different objects. Although the bandwidth has been increasing exponentially in the last few years, the web experience is not improving at the same pace because of latency issues in HTTP/1. The HTTP/2 protocol aims to solve these issues by allowing clients and servers to multiplex HTTP requests and responses on a single TCP connection. If HTTP/2 is widely adopted, it can have enormous benefits not only for the user experience, but also for the servers and the network. Since clients do not have to open multiple parallel connections to avoid the problem of head-of-line blocking in HTTP/1.1, the number of concurrent TCP sessions can be significantly reduced.**

However, although multiplexing is one of the main features of HTTP/2, nothing actually prevents a client from opening multiple HTTP/2 connections to a server. In this paper we investigate the behavior of HTTP/2 traffic in the wild. We perform experiments to examine if web browsers use a single connection per domain over HTTP/2 in practice. Contrary to popular belief, our experiments on the traffic of a large university campus network and a residential network show that a significant number of HTTP/2 accesses are performed using parallel connections to a single domain on a server. We present two possible hypotheses for this behavior and discuss its implications for the future of the web.

I. Introduction

The Hypertext Transfer Protocol (HTTP) is the most popular protocol for delivering web content. In the last two decades the web content has evolved from simple text and images to online gaming, video conferencing, music streaming, social networking etc and its complexity is constantly increasing. According to [1] the average page size for the Alexa top 1000 websites in 2016 has reached 2.1 MB and fetching a page requires requests to 128 different objects. Latency is another major issue faced by modern web services. Even though the bandwidth has been increasing exponentially in the last few years, the web experience is not improving at the same pace because of latency issues in HTTP. This triggered a major update to the HTTP protocol. In 2009 Google developed the SPDY protocol, which introduced several features to address the limitations of HTTP/1.1. It gained a lot of popularity and was deployed on some of the most popular websites like Google, Facebook and Twitter. In 2015, the Internet Engineering Task Force (IETF) specified the HTTP/2 [2] protocol which is heavily based on SPDY and aims at increasing the performance of modern web applications. As of May 2016, more than 120K domains support HTTP/2 compared to just over 600 in May 2015 [3]. HTTP/2 is supported by all major PC and mobile browsers including Chrome, Firefox and iOS Safari, and popular web servers such as Apache, nginx, OpenGSE and LightSpeed [3], [4]. It is evident that HTTP/2 is gaining popularity and drawing great attention from the web industry.

A. Motivation

The much awaited update to HTTP is officially complete and is drawing great interest from web industry and research community. HTTP/2, like its predecessor SPDY, allows the client and server to multiplex HTTP requests and responses on one single TCP connection, while avoiding the problem of head-of-line blocking of HTTP/1.1. In this way, one can expect that a wide deployment of HTTP/2 will not only improve the latency of web services, but also bring great benefits for web servers due to reduction of concurrent TCP sessions. Similarly, it would benefit network operators thanks to the reduced resource usage in firewalls, NAT tables, or flow monitors.

However, although multiplexing requests over a single TCP connection is one of the main features of HTTP/2, nothing actually prevents a client from opening *multiple* HTTP/2 connections to a server. We have seen in the past that web browsers do not always follow the original design of protocols. Instead, they implement the most optimal solution with respect to performance. This motivates us to study empirical HTTP/2 traffic and examine how connections are used in practice.

B. Contribution

We study the behavior of HTTP/2 and SPDY clients when establishing and maintaining connections to web servers. To this end, we collect traffic data in two different networks and detect concurrent connections established by the clients.

We show that clients indeed open multiple HTTP/2 and SPDY connections to servers and that they actively use those connections. We also show that the number of concurrently opened connections vary with the type of web application. Considering the fact that prior studies [5], [6], [7] have evaluated the performance of HTTP/2 under the assumption that there is always a single connection, we believe that this new insight into the real usage of HTTP/2 will lead to more

978-3-901882-85-2 © 2016 IFIP

accurate traffic models for planning and management tasks.

Our paper is structured as follows. In Section II, we compare HTTP/1 and HTTP/2 and discuss related work. We describe our methodology and data collection procedure in Section III. The results of our analysis are presented and discussed in Section IV. The paper is concluded in Section V.

II. Background and Related Work

HTTP is an application-level protocol developed in the early nineties for delivering documents and other web resources over the Internet [8]. As the web content evolved during the last two decades from simple text documents to complex web pages consisting of hundreds of resources, web browsers started creating multiple connections to the server to fetch these resources in parallel to increase performance. Modern browsers create up to 13 connections per domain name [9]. On the server side *domain sharding* is used which further increase the overall number of connections between a client and server. This results in an increase in server load and network congestion. HTTP/2 solves the above mentioned issues of HTTP/1.1 by introducing multiplexing and also offers several other new features. The evolution of the HTTP protocol is summarized in Figure 1.

A. *HTTP/2*

HTTP/2 enables a more efficient use of network resources and reduces latency by introducing several new features. We discuss the most important ones next.

- Multiplexing: One of the most important features of HTTP/2 is multiplexing. It allows clients and servers to send multiple HTTP requests and responses asynchronously in multiple streams on a single TCP connection. Responses can be interleaved, avoiding in this way head-of-line blocking of HTTP/1.1.
- Compression: The header data is compressed to reduce the overheads of redundant header fields. HPACK [10] is used for compression.
- Server Push: It allows the server to proactively push resources to clients. When a client requests a resource, the server predicts the other resources related to it, and pushes them along with the response of the original request. This helps improving the page load time.
- Content priority: Some resources are more important than others when rendering a page. A client is allowed to specify the importance of each resource, so that it can be transferred by the server in the preferred order.

B. *Related work*

We study the behavior of HTTP/2 and SPDY in the wild. HTTP/2 is relatively new but its predecessor SPDY [11] has been used for several years now and there are a number of studies on its performance and on its comparison with HTTP/1. Many important features of HTTP/2, such as multiplexing, compression, server push and content priority were already present in SPDY.

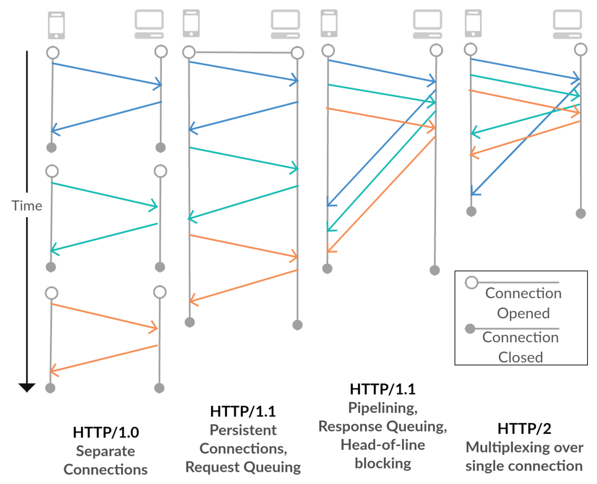
Time
Connection Opened
Connection Closed
HTTP/1.0 Separate Connections
HTTP/1.1 Persistent Connections, Request Queuing
HTTP/1.1 Pipelining, Response Queuing, Head-of-line blocking
HTTP/2 Multiplexing over single connection

Fig. 1. Comparison of HTTP versions

In [6] the authors identify the impact of network characteristics and website infrastructure on the performance of SPDY. They conclude that SPDY's default use of a single connection might be unwise in high bandwidth networks because it leads to lower performance compared to HTTP/1, which can exploit higher throughput by opening parallel connections. The authors of [5] perform extensive tests with both synthetic and real pages. They find that when there is little network loss, SPDY has higher performance, but in high loss scenarios the single connection hurts performance. In [7] the authors perform experiments on HTTP/2 traffic focusing on cellular networks. They draw similar conclusions as in [5], i.e., HTTP/2 performance is reduced in cellular networks because packet losses negatively affect HTTP/2 performance due to the usage of a single TCP connection. In contrast, our experiments show that the usage of a single connection in SPDY and HTTP/2 is not always true and the browsers create several parallel connections using these protocols in a significant number of cases.

In [12] the authors compare SPDY's single long *elephant* flow to highly concurrent short-lived HTTP *mice* flows. They also point out that SPDY's single-connection approach has a disadvantage because of inequitable TCP backoff compared to HTTP/1, which uses multiple TCP connections. The authors discourage connection proliferation, but recommend using a small number of concurrent SPDY connections as a short-term mitigation strategy. In this paper we show that this strategy is in fact implemented by the major browsers supporting SPDY and HTTP/2. Thus, ignoring this fact can lead to wrong interpretations of SPDY and HTTP/2 performance evaluations.

Perhaps the most closely related work to ours is [13], in which the authors do large-scale experiments to check the adoption of HTTP/2 and test its performance. They observe that HTTP/2 does not currently manage to serve a page using a single TCP connection. They show that half of the websites using HTTP/2 today use at least 20 TCP connections. The reason is that most websites have implemented HTTP/1 optimization techniques like domain sharding (splitting resources across multiple domains), which causes browsers to create a separate TCP connection for each domain. In our work we show that even if domain sharding is removed, the browsers still create multiple connections even to the *same* domain using SPDY and HTTP/2.

III. Datasets and Methodology

A. Datasets

We rely on Tstat [14] to perform passive measurements and collect data related to SPDY and HTTP/2 usage in different networks. Tstat monitors each TCP connection, exposing flow level information,[1] including: (i) client and server IP addresses, (ii) timestamps of the first and the last packets in each flow, (iii) the amount of exchanged data, (iv) Server Name Identification (SNI) strings found in TLS handshakes, and (v) the Fully Qualified Domain Name (FQDN) that the client resolved via DNS queries prior to opening connections [15]. The latter two fields are instrumental to classify traffic, since they expose names of websites contacted by the users. We explore those fields to characterize the usage of parallel connections by SPDY and HTTP/2 considering different services.

In addition, Tstat classifies traffic per protocol by means of a typical Deep Packet Inspection (DPI) approach. Specifically, SPDY and HTTP/2 flows are identified by inspecting flags in TLS handshakes. Tstat explores NPN/ALPN fields in initial TLS messages. The Next Protocol Negotiation (NPN) and Application-Layer Protocol Negotiation (ALPN) [16] are TLS extensions that allows clients and servers to negotiate the application protocol to be used in TLS connections. The client and server advertise their supported protocols in its *Client Hello* and *Server Hello* messages. A suitable protocol that is common between the two is finally selected. Tstat exports this list of protocols advertised in the TLS handshake, thus allowing us to check the possible protocols used in the connections.

We have collected traffic with Tstat in two different networks. Table I summarizes our datasets. The Campus dataset has been collected at border routers of a campus network. It includes traffic of around $15,000$ users in a European university. The dataset covers wired workstations in research and administrative offices as well as WiFi access points. The Residential dataset has been collected at a Point of Presence (PoP) of a European ISP, aggregating around $25,000$ households. ADSL and Fiber To The Home (FTTH) provide access to the users. Ethernet and/or WiFi are used at home to connect the end users' devices, such as PCs and smartphones.

By the time of writing, the version of Tstat deployed in the evaluated networks is only capable of identifying SPDY traffic and HTTP/2 traffic up to its draft version 14. Thus, we miss deployments relying on the final HTTP/2 version standardized by the IETF in 2015.

[1]We adopt the classic definition for a flow, which is also implemented by Tstat – i.e., client and server IP addresses, client and server port numbers and transport layer protocol define a flow.

TABLE I
Summary of datasets (Jan-Apr 2016)

Dataset		Campus	Residential
SPDY	Flows	38 million	160 million
	Bytes	7.3 TB	31 TB
	FQDNs	66,952	268,827
	2nd Level Domains	6,207	8,399
HTTP/2	Flows	4 million	21 million
	Bytes	1.3 TB	6 TB
	FQDNs	2,432	4,663
	2nd Level Domains	208	298

In total, we have observed 198 million flows where SPDY was negotiated, as well as 25 million HTTP/2 flows, during 4 months of data collection (Jan-Apr 2016). Table I lists the number of server FQDNs contacted by means of each protocol. We can see that users in the monitored networks reach more than 5 k (250 k) FQDNs using HTTP/2 (SPDY), or 200 (8,000) names when considering only second-level domains. Both protocols together account for more than 45 TB of traffic. Thus, our datasets provide a large-scale view in terms of number of users and services.

Finally, in the rest of the paper we will always evaluate SPDY and HTTP/2 flows together, since both protocols have the core feature of multiplexing and we want to know whether they open a single connection with a server. We will however look only at those flows transported over TCP/TLS, ignoring for instance QUIC and non-encrypted HTTP/2, since the former is by far more common in current traffic. UDP makes around 15-20% of the total incoming IP traffic in one of the probes where we collected data (ADSL). Up to 50-60% of the UDP traffic is QUIC in this probe. In the other probe, where we have ADSL and FTTH, P2P is more common and QUIC's share is only 5% of the overall incoming IP traffic.

B. Methodology

Our goal is to quantify how many flows are opened in *parallel* by clients when communicating with a server. Therefore, we derive a methodology to estimate the number of parallel flows. We define parallel flows as those cases where at least two flows (i) have the same client and server IP addresses, (ii) share the *domain name* coming from SNI strings, (iii) present different client port numbers, and (iv) start close in time. The last requirement has been imposed to reduce the possible influence of NATs, as we will discuss next.

We apply a simple method to mark flows as *isolated* or *parallel* in our traces. We first filter the traces to select only flows where HTTP/2 or SPDY was negotiated. After that, we aggregate flows using the following algorithm.

Our algorithm reads a list of flows sorted by start time. It maintains a set of active "sessions" between clients and servers, where a session is identified by *client IP address*, *server IP address*, *domain name* and *start time*. The latter is the start time of the first flow of the session. When a flow f is read, the algorithm checks whether its fields *client IP address*, *server IP address* and *domain name* match an active session. If such a session exists and the start time of f is no later than the start time of the session plus a fixed window size `MAX_GAP`, the flow f is added to that session. Otherwise, the old session (if any) is closed and a new session is started with f as its first flow.

The result of our algorithm is a list of all identified sessions, i.e., sets of flows with identical *client IP address*, *server IP address*, and *domain name* where no flow starts later than `MAX_GAP` time units after the earliest flow in that set. We call a flow *isolated* if it is the only flow in its session, otherwise it is a *parallel* flow.

C. Aggregation gap and impact of NAT

Our analysis could be affected by NATs. That is, when a large number of users are connected from a single IP address, the probability that different users will simultaneously contact the same server increases. Connections of the different users would therefore be marked as parallel by our algorithm.

As a first step, we manually remove IP addresses of large NATs in the Campus dataset, based on our knowledge of the campus network topology. For instance, some IP addresses that we know to host WiFi access points relying on NATs are manually removed from the analysis. This step considers around 12.5 million flows (i.e., $\approx$30% of the raw dataset).

In order to estimate the probability that two users behind a NAT contact the same server at the time (or, at least, close enough that our algorithm would assign their flows to the same session), we examine the User Agents found in our datasets. User Agents are strings containing information about the client browser, which are sent out by clients when contacting URLs using the HTTP protocol. We leverage the HTTP monitoring plug-in of Tstat [14] to export information present in unencrypted HTTP/1 headers for every HTTP request seen in the networks. This collection of User Agents is performed simultaneously to our flow-level captures.

We count how often different User Agents are observed per IP address by dividing our data traces in time bins of 1 min. We have found that 90% of those time bins have only 4 or less different User Agents. Collisions are even rarer: In less 1% of the time bins two User Agents access the same server (via unencrypted HTTP/1).

Note that the User Agent is *not* a unique identification for a user. It however contains many details, such as the browser and operating system versions. Since our datasets contain only small NATs, e.g., aggregating few users' devices in households, we believe that the odds that different users rely on exactly the same browser configuration and contact a single server simultaneously are negligible.

We therefore adopt a gap of 5 seconds for our algorithm in the experiments that follow, since it is a reasonable value for browsers to open parallel connections when loading a web page. Compared to the 1-minute bin size studied above, this value is a rather conservative choice and it is likely that we under-estimate the real number of parallel connections.

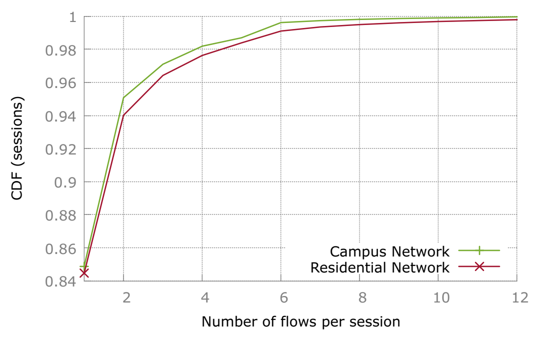

Fig. 2. CDF of number of flows per session

TABLE II
SUMMARY OF PARALLEL HTTP2 AND SPDY FLOWS

Data set	Campus Network	Residential Network
Ratio of isolated flows	67.1%	65.3%
Ratio of parallel flows	32.9%	34.7%

IV. RESULTS

A. Observed parallel flows

We have applied our algorithm to the datasets of the two monitored networks. Table II gives a summary of the results.

For both campus and residential networks, we observe that around two thirds of the studied HTTP/2 and SPDY flows are isolated, i.e., the algorithm could not identify parallel flows for them according to the rules described in Section III-B. In contrast, one third of the flows are parallel flows which comprise around 15% of sessions. Sessions with parallel flows contain around 3 flows on average. Figure 2 shows the CDF of the number of flows per session.

There are several reasons for the large number of sessions with only one (isolated) flow. One reason could be the fact that parallel connections to a server are mostly created in the beginning of loading a web page to fetch resources like html, images, css, javascript etc. Once all the required resources are in browser's cache, the subsequent interactions of the client with the website are usually served by creating a single connection from time to time.

As already stated, one of the main advantages of HTTP/2 and SPDY is its capability to multiplex multiple requests and responses over one single TCP connection. By opening multiple connections to a server, a client is approaching a communication pattern similar to HTTP/1.1. Therefore, the results in Table II raise a question about the possible reasons for this behavior. We have discussed our finding with experts from the industry associated with Google Chrome, Mozilla and the IETF HTTP working group. Their responses allow two possible explanations.

B. Potential reasons for parallel flows

a) Hypothesis 1: When a browser is oblivious to the protocols supported by the server, it prepares a number (typically 6) of connections that it could use for HTTP/1 just in case if the server does not support HTTP/2 or SPDY. Later on when the browser discovers that the server speaks HTTP/2 or SPDY, it closes the superfluous connections. Exactly how the browser does this may differ between various implementations but this should be done rather quickly so that the extra connections don't consume resources for long.

b) Hypothesis 2: Although HTTP/2 and SPDY allows to multiplex all requests to a server on one single connection, a web browser still creates multiple connections for performance reasons. Unlike in hypothesis 1, these connections are actively used and not closed quickly. By doing this, the browser avoids the performance problems that might occur with single connections in scenarios with high packet losses (see Section II-B). Furthermore, in some situations the TCP congestion window (CWND) negatively affects the performance because it places limits on the amount of unacknowledged data which can be in transit between two endpoints at a given time on a single connection. Therefore, the browser chooses to use more connections to avoid performance bottlenecks.

We verify these hypotheses in the following sections.

C. Degree of time-overlap of parallel flows

Let $\{f_1, \ldots, f_n\}$ be a session consisting of $n > 1$ parallel flows between a client and a server, as identified by our algorithm. If hypothesis 1 is correct, we expect that a session mainly consists of one long flow and several shorter flows (corresponding to the superfluous connections that are closed once the server has been identified as HTTP/2 or SPDY-capable). To verify this behavior, we select the longest flow f_l of the session and the flow f_o in the session that overlaps for the longest duration with f_l. We calculate the percentage p of overlap by

$$p = \frac{duration\ of\ overlap}{duration\ of\ f_l} \times 100$$

For hypothesis 1, we expect that the sessions in our datasets have a small overlap, while a high overlap would support hypothesis 2.

Figure 3 shows the CDF of the degree of overlap for all sessions with parallel flows in our datasets. We can see that in 50% of those sessions, the percentage of overlap is at least 75% in the campus network and 63% in the residential network. Furthermore, in more than 20% of the cases in the campus network and 30% in the residential network, there is nearly complete overlap. In absolute numbers, the average overlap duration in the campus network and the residential network is 58.5 seconds and 29 seconds, respectively. This suggests that many sessions have at least two parallel connections that are not quickly closed by the browser.

D. Degree of time-overlap vs flow duration

The previous experiment shows a large degree of overlap of parallel flows but it does not explain why. One could argue that most flows are extremely short and hence overlap or, quite the opposite, most overlapping flows are just very long flows that the browser keeps alive and never close.

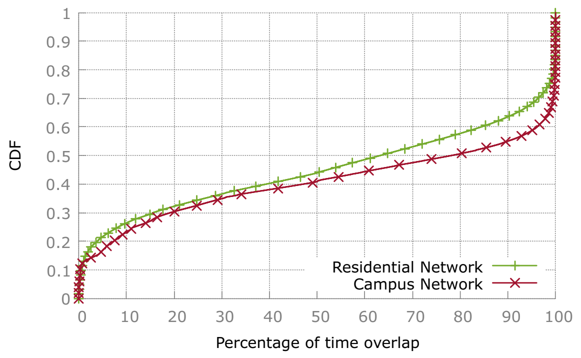

Fig. 3. CDF of degree of time overlap

To get a better understanding of the data, we have divided the sessions according to the length of the longest flow f_l in four categories.

In Figure 4 and 5, we show again the CDF of the percentage of overlap, this time with the sessions classified according to the duration of their longest flow.

In both networks, we can see that sessions with duration > 60 s tend to overlap for significantly less time compared to shorter sessions. This speaks against the theory that long parallel flows with high overlap are "forgotten" flows. A large fraction of sessions with duration < 10 s have a high overlap but we also see a comparably high overlap for sessions having duration in the range of 10 s to 600 s.

E. Degree of data-overlap of parallel flows

Although we have shown in the previous experiments that clients open parallel connections with overlapping life times, it does not necessarily mean that the client and the server are actively using these connections for data transmission. To measure how the data is distributed over parallel connections inside sessions with at least two flows, we calculate for each session the ratio

$$r = \frac{total\ session\ size - size\ of\ f_s}{size\ of\ f_s} \times 100$$

where f_s is the largest flow of the session (in number of bytes) and the total session size is the sum of the sizes of all flows in the session. A ratio of 100% would mean that the largest flow transports the same amount of bytes as all other parallel flows combined. Larger ratios would mean that the largest flow does not carry the majority of bytes.

We show in Figures 6 and 7 the CDF of r for the sessions in the two networks. We again create different intervals to highlight the behavior of small and large sessions. We split the data into five groups, based on the size of the largest flow in the session. For the residential (campus) network, 11% (18%) of the sessions fall in the category < 5 kB, 21% (30%) in the category 5–10 kB, 32% (25%) in the category 10–50 kB, 27% (22%) in the category 50–500 kB, and 8% (6%) in the category > 500 kB.

We see a significant number of sessions where bytes are spread over multiple connections. Notice, for instance, that

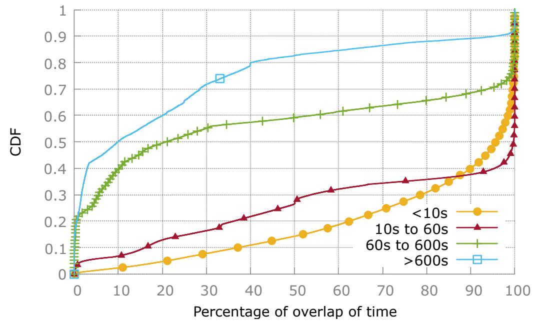

Fig. 4. CDF of degree of time-overlap for flows of different lengths in campus traffic

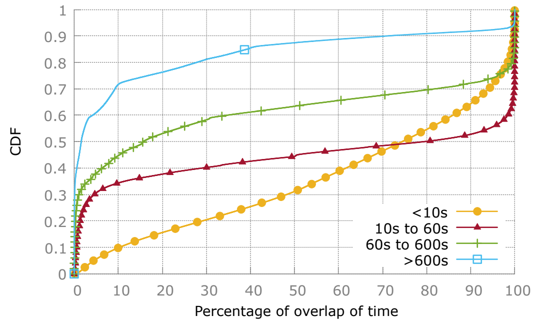

Fig. 5. CDF of degree of time-overlap for flows of different lengths in residential traffic

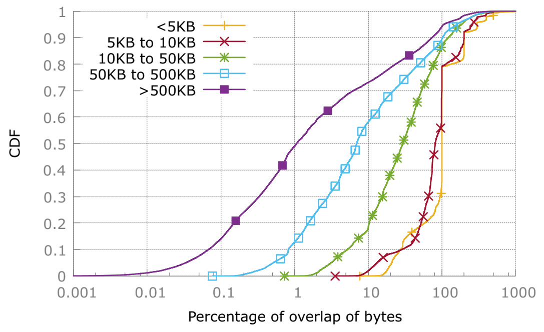

Fig. 6. CDF of data-overlap ratio in campus traffic

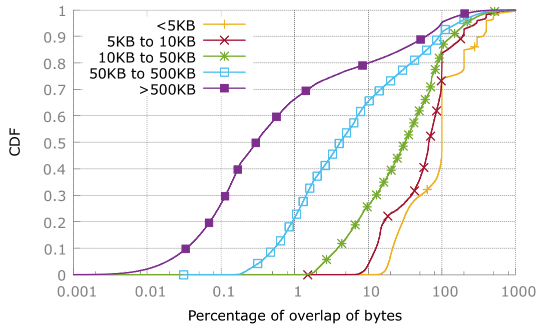

Fig. 7. CDF of data-overlap ratio in residential traffic

for around 20% of the sessions where the largest flow carries between 10–50 kB we have a ratio $r \geq 100\%$. That means that the remaining connections carry at least the same amount of bytes as the largest connection.

For very small sessions, i.e., largest flow carrying less than 5 kB, we see that the ratio r is very concentrated around 100%. This is not surprising since those sessions mostly contain TLS handshakes and barely any payload. On the other extreme, very large sessions seem to concentrate more bytes in the largest connection. However, even for those cases, we still see a non-negligible number of sessions where bytes are spread over multiple connections.

This suggests that the parallel flows are actively used by the browser, thus hinting that our second hypothesis is true.

V. Conclusion

This paper studied the behavior of clients when opening SPDY and HTTP/2 connections in the wild. The HTTP/2 and SPDY specification advises browsers to open a single connection and multiplex requests, but our experiments show that around 1/3rd of all connections of these protocols are parallel connections with the arithmetic mean value of 3 and median of 2 connections per domain. Our data does not allow us to precisely pinpoint the reasons behind this behavior. Yet, results indicate that this design choice is motivated by performance, with browsers actively using the parallel connections to exchange data, e.g., to avoid the performance penalties that a single TCP connection would bring in case of packet loss.

Our findings are very important for future studies on HTTP/2 performance. Previous works have argued that HTTP/2 faces performance degradation in some environments due to the use of a single connection, when compared to the many connections used by HTTP/1. We showed that browser implementers have taken measures to avoid these limitations. Researchers evaluating HTTP/2 should, therefore, be careful in assuming that implementations follow RFCs' recommendations, even if the protocol standardization is rather recent.

Regarding more general implications for the Internet, HTTP/2 comes with a promise of reducing the number of connections between clients and servers. This would have a positive impact on servers and on network infrastructure. For instance, it could reduce the workload on devices such as routers, firewalls and intrusion detection systems, which usually keep state per TCP connection in the network. However, our results showed that this promise is rather far from being realized. Although prior studies have shown that there is a reduction in number of connections in HTTP/2 when compared to HTTP/1, we are still far from the point where only one connection will be required to load a web page.

Acknowledgment

This work was partially supported by the Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC) funded by the European Commission (EACEA) under the FPA 2012-0030. We thank Mike Belshe, Mark Nottingham and Daniel Stenberg for their valuable input.

References

[1] The http archive. [Online]. Available: http://httparchive.org/trends.php

[2] M. Belsche, R. Peon, and M. Thomson. (2015) Request for comments 7540 - hypertext transfer protocol version 2 (http/2). [Online]. Available: https://tools.ietf.org/html/rfc7540

[3] Http/2 adoption. [Online]. Available: http://isthewebhttp2yet.com/measurements/adoption.html

[4] Browser support tables. [Online]. Available: http://caniuse.com/

[5] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How speedy is spdy?" in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 387–399.

[6] Y. Elkhatib, G. Tyson, and M. Welzl, "Can spdy really make the web faster?" in *Networking Conference, 2014 IFIP*, June 2014, pp. 1–9.

[7] H. de Saxcé, I. Oprescu, and Y. Chen, "Is http/2 really faster than http/1.1?" in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2015, pp. 293–299.

[8] T. Berners-Lee, R. Fielding, and H. Frystyk. (1996) Requests for comments 1945 - hypertext transfer protocol – http/1.0. [Online]. Available: https://www.ietf.org/rfc/rfc1945

[9] Web browsers profiling. [Online]. Available: http://www.browserscope.org

[10] R. Peon and H. Ruellan. (2015) Request for comments 7541 - hpack: Header compression for http/2. [Online]. Available: https://tools.ietf.org/rfc/rfc7541.txt

[11] Spdy. [Online]. Available: https://www.chromium.org/spdy

[12] B. Thomas, R. Jurdak, and I. Atkinson, "Spdying up the web," *Commun. ACM*, vol. 55, no. 12, pp. 64–73, Dec. 2012.

[13] M. Varvello, K. Schomp, D. Naylor, J. Blackburn, A. Finamore, and K. Papagiannaki, *Is the Web HTTP/2 Yet?*, 2016, ch. Passive and Active Measurement: 17th International Conference, PAM 2016, Heraklion, Greece, March 31 - April 1, 2016. Proceedings, pp. 218–232.

[14] A. Finamore, M. Mellia, M. Meo, M. M. Munafò, and D. Rossi, "Experiences of Internet Traffic Monitoring with Tstat," *IEEE Network*, vol. 25, no. 3, pp. 8–14, 2011.

[15] I. Bermudez, M. Mellia, M. M. Munafò, R. Keralapura, and A. Nucci, "DNS to the Rescue: Discerning Content and Services in a Tangled Web," in *Proc. of the ACM Internet Measurement Conference*, 2012, pp. 413–426.

[16] S. Friedl, A. Popov, A. Langley, and E. Stephan. (2014) Transport layer security (tls) application-layer protocol negotiation extension. RFC 7301 (Proposed Standard). Internet Engineering Task Force. [Online]. Available: http://www.ietf.org/rfc/rfc7301.txt