Evaluating the Performance of Next Generation Web Access via Satellite

Raffaello Secchi(✉), Althaff Mohideen, and Gorry Fairhurst

School of Engineering, University of Aberdeen,
Fraser Noble Building, Aberdeen AB24 3UE, UK
r.secchi@abdn.ac.uk

Abstract. Responsiveness is a critical metric for web performance. Update to the web protocols to reduce web page latency have been introduced by joint work between the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C). This has resulted in new protocols, including HTTP/2 and TCP modifications, offering an alternative to the hypertext transfer protocol (HTTP/1.1). This paper evaluates the performance of the new web architecture over an operational satellite network. It presents the main features of the new protocols and discusses their impact when using a satellite network. Our tests comparing the performance of web-based applications over the satellite network with HTTP/2 confirm important reductions of page load times with respect to HTTP/1.1. However, it was also shown that performance could be significantly improved by changing the default server/client HTTP/2 configuration to best suit the satellite network.

Keywords: SPDY · HTTP/2 · PEP

1 Introduction

In the last ten years, the types of data exchanged using HTTP has changed radically. Early web pages typically consisted of a few tens of kilobytes of data and were normally not updated frequently (static web). Today, typical web pages are more complex, consisting of (many) tens of elements, including images, style sheets, programming scripts, audio/video clips, HTML frames, etc.[1,2]. Moreover, many web pages are updated in real time (e.g., when linked to live events or dynamic databases). Web interfaces have found use in a wide range of non-browsing applications, e.g., to interact with distributed web applications, where both data and application reside at a remote side.

Raffaello Secchi was funded by the European Community under its 7th Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700).

Althaff Mohideen was by the RCUK Digital Economy programme of the dot.rural Digital Economy Hub; award reference: EP/G066051/1.

P. Pillai et al. (Eds.): WiSATS 2015, LNICST 154, pp. 163–176, 2015.
DOI: 10.1007/978-3-319-25479-1_12

The HTTP/1.x suite of protocols [3,4] has dominated web usage for over twenty years. It has become widely used in a variety of contexts beyond its original goals, limitations of the HTTP protocols have become apparent. The original HTTP request/response model, where each web object was requested separately from the others, has been shown to not scale to modern complex web pages and can introduces a significant amount of overhead at start-up. HTTP interacts poorly with the transport layer (Transmission Control Protocol [5]) resulting in poor responsiveness for the web. These performance issues have been accompanied by changing patterns of web.

Changes in the size and structure of web pages now mean that HTTP/1.x can become a significant bottleneck in web performance [6]. HTTP/1.0 [3] allowed only one request at a time on a given TCP connection. HTTP/1.1 [4] added HTTP persistence, i.e., the ability to reuse a TCP connection for subsequent HTTP object requests. HTTP persistence feature avoided the overhead of the TCP connection establishment, but did not resolve the performance gap. The growing number of objects per page, caused web designers to increase the number concurrent TCP connections to decrease down-load time, sometimes scattering the components of a web page over many servers, known as "sharding". While opening many TCP connections can reduce page load times, it is not a network-friendly solution, because it results in web sessions competing aggressively with other network flows. It also does not resolve the head-of-line blocking (HoLB) that occurs when an important object is scheduled over a slow TCP connection and received after a less important one.

Initiatives from Google^{TM}proposed new protocols as an alternative to HTTP 1.1, better suited to new web content, and designed to make HTTP more responsive and more network-friendly. SPDY [7] and later QUIC [8] are prototype protocol implementations of this new approach. In 2009, work on SPDY triggered formation of the HTTPbis working group in the Internet Engineering Task Force (IETF) and related work in Worldwide Web Consortium (W3C). This has since resulted in the definition of a new standard, HTTP/2 [9] in 2015.

This paper seeks to understand the implications for the satellite community of the introduction of HTTP/2 and other TCP/IP stack modifications. We show that HTTP/2 offers significant benefits in terms of reduction of the page load time (PLT) in different scenarios, including cases with performance enhancement proxies (PEPs) and encrypted tunnels. However, we also observed that the default HTTP/2 configuration is optimised for a terrestrial networking and may not be suitable for the satellite scenario, thus negatively impacting HTTP/2 performance. This suggests that HTTP/2 may not be yet mature to support high latency links.

This conclusion is also backed by previous studies that considered the performance of HTTP/2 over terrestrial and satellite networks. An extensive comparison of HTTP(S) and HTTP/2 in terrestrial networks was presented in [6]. This study confirms that HTTP/2 effectively address the problems of HTTP but also highlights some of its potential pitfalls. In particular, the paper observes that a single connection is more exposed to transient loss episodes than multiple HTTP/1.1 connections, and that the HTTP/2 could under-perform due

to the current highly *sharded* structure of the web. Performance limitations of HTTP/2 were also pointed out in [10] where PLT was related to the structure of the page. Tests with HTTP/2 over satellite were performed in [11–13]. In [11] it was observed that a proper scheduling discipline and countermeasure to compensate for wireless errors may be needed to achieve good link utilisation with HTTP/2. In [13] HTTP/2 performance were compared to HTTP/1.1 with various demand/assignment multiple access (DAMA) schemes concluding that HTTP/2 can achieve in some scenarios close-to-terrestrial performance.

The remainder of this paper is organised in a series of sections. Section 2 introduces the novelties of HTTP/2 and discusses their usage over satellite. Section 3 surveys the new TCP modification proposals to make it more responsive for the new type of web applications. Section 4 reports the results of the experiments over the satellite platform. A conclusion on the results and the new web architecture is given in Sect. 5.

2 A New Architecture for Web Technology

HTTP/2 maps the entire bundle of HTTP request/response transactions required to retrieve a web page onto a single TCP connection. It introduces the concept of *stream* as an independent data flow within the TCP connection. This allows interleaving of *frames* (stream protocol data units) onto the connection and the prioritisation of streams, letting more important requests complete faster. In addition, HTTP/2 defines a Server Push mechanism, an experimental communication mode where the server can send data to a client without an explicit request. Server Push enables a fully bidirectional client-server channel and may be used by a server to anticipate client requests predicting what data the client is going to request.

2.1 Connection Setup

Although HTTP/2 is designed to replace HTTP/1.x, an initial deployment phase is envisaged where HTTP/2 will co-exist with older HTTP/1.x systems (The httpbis working group therefore also introduced a number of innovations into HTTP/1.1.) Not all servers will immediately change to use HTTP/2, and coexistence with HTTP/1.1 is expected for the foreseeable future.

The design of HTTP/2 chose to reuse the same URI scheme of HTTP and HTTPS to indicate HTTP/2 web resources. The new protocols also use the same port numbers (TCP port 80 and 443). This required a method to signal use of HTTP/2. HTTP/1.1 defined the *Switching Protocol* header, that, when inserted in a request, indicates the intention of the client to change the session protocol. This mechanism, however, costs an extra RTT after connection establishment. If HTTP uses SSL, however, the application-layer protocol negotiation (ALPN) transport-layer security (TLS) extension header could be used to signal the presence of HTTP/2 during the TLS handshake.

ALPN was the preferred method when the HTTP/2 connection is made over SSL, expected to be used in the majority of cases with HTTP/2. Use of SSL also avoid middle-boxes hindrance as the new protocol is deployed. Using SSL requires a SSL handshake before every HTTP/2 session can be established.

2.2 Multiplexing

HTTP/2 allows a session to multiplex transmission of web page resources using a Stream. Streams are independent, bi-directional sequences of frames between the server and the client. Streams can be initiated by both a client and a server. (In HTTP/1.1, the HTTP connection could only be started by the client). A HTTP/2 connection can contain multiple concurrently open streams, with either end-point interleaving frames from multiple streams. This removes the need to open multiple transport connections.

Multiplexing resources can offer important performance benefits for long delay paths, including clients that access the network via satellite. This reduces the amount of traffic over the network removing the overhead of opening several TCP connections and allows the server to send more data per TCP segment. Multiplexing over a single connection could also reduce the need to place HTTP proxies at the edge of a satellite network (i.e., reducing the need for application-layer protocol enhancing proxies - PEPs).

2.3 Flow Control

Flow Control has been introduced in HTTP/2 to regulate contention among the streams that share a TCP connection. This avoids, for instance, a stream from being blocked by another contending stream. Flow Control also seeks to regulate the overall connection rate. This can be used to protect resource-constrained endpoints from being overwhelmed by received data. This addresses cases where the receiver is unable to process data on one stream, yet wants to continue to process other streams within the same connection. In the case of a multi-hop HTTP/2 connection, each hop receiver can control the rate over its hop.

HTTP/2 uses a simple credit-based flow control mechanism. Clients are required to specify an initial size in the SETTINGS frame indicating the amount of data they are prepared to accept for the entire connection and for each stream. During the session, clients use WINDOW_UPDATE frames to increase the size of a stream or the connection. The WINDOW_UPDATE frame specifies the increment in size of the stream.

While Flow Control is an essential feature of HTTP/2, the Flow Control mechanisms are still at an early development stage. The current standard only provides the necessary tools to implement this feature, but does not mandate any particular method.

The choice of the size of the streams can have a direct impact on the performance of the long-delay paths found with satellite networks. Too small values in initial settings may be insufficient to achieve high throughput over a large bandwidth-delay product connection.

2.4 Server Push

HTTP/2 allows a server to *push* responses to a client associated with a previous client request. This could be useful when the server can anticipate which resources the client is going to request based on previous requests. To avoid *pushing* undesired data a client can request that Server Push is disabled when the connection settings are negotiated.

Server Push is implemented using the PUSH_PROMISE frame, which contains the set of requests that would originate a response. The PUSH_PROMISE frame is followed by PUSH_RESPONSE frames that contains the actual pushed data. If the client decides to accept the pushed data, it is required not to send any request for the pushed resources. Otherwise, the client can issue a control frame to stop the transmission of the data.

Server Push could be beneficial for users connected via satellite, because it could potentially reduce the time to load a page by at least one RTT. For example, eliminating the need for a client to send a request for the resources linked to the index page. However, the optimisation may come at a cost, since the client is less able to decide what content uses the satellite capacity. A similar concern has also been expressed by mobile cellular operators, where pro-actively downloading non-needed contents can incur into additional cost [10].

3 Transport Layer Modifications

The effectiveness of HTTP modifications would be limited if not complemented by changes at the transport layer. The awareness that the transport layer needs to be harmonised to the web application layer for higher responsiveness and lower overhead has provoked a series of proposals since 2010. The new protocols update the way web protocols use the Internet by using new transport mechanisms (larger initial window, initial data, updates to congestion window validation, support for thin flows, etc.).

Although most TCP modifications were designed to improve performance over terrestrial networks, they can impact the performance for satellite networks. Removing even a few initial RTTs has macroscopic benefits for the satellite networking. This section surveys key recent proposals to update TCP.

3.1 Fast Open

TCP currently only permits transmission of useful data after the two endpoints have established a connection [5]. This initial three-way handshake (3WHS) introduces one RTT of delay for each connection. For short data transfers, the extra RTT incurs a significant portion of the flow duration [14]. According to [15] the cost of the initial handshake is between 10 % and 30 % of the latency for a HTTP transaction. Data on SYN was initially proposed in RFC793, but later discouraged because this would allow data to be delivered to the application before the connection has been established.

TCP Fast Open (TFO) [16] is an IETF experimental specification published in December 2014 that provides new rules that allow data to be carried in an initial SYN segment and to be consumed by the receiving endpoint during the initial connection handshake. This can save one RTT compared to a standard TCP connection.

In TFO, the server side uses a new security mechanism (based on cookies) to authenticate a client initiating a connection thus addressing previous data integrity concerns caused by dubious SYN packets. This avoids the pitfalls of earlier methods, such as T/TCP [17].

TFO is designed to deliver data safely during the 3WHS without requiring the server to store per-client authentication information. A client obtains a TFO cookie from the server when access the server for the first time. This cookie contains encrypted information that authenticates the client (for example, the client's IP address). When the client wants to open a new connection to the same server, it returns the cookie to the server by inserting it into the payload. The server recognizes the IP address of the client deciphering the cookie, and therefore it can accept the TCP payload of the SYN segment.

TFO can reduce the delay of initial connection establishment, which is non negligible in the overall web page download delay budget as shown in the next section.

3.2 Larger Initial Window

RFC 3390 specifies the TCP initial window to be 3 segments [18]. This has been widely deployed. Motivated by the desire to improve Fast Retransmit, Google proposed further increasing the IW to 10 segments (about 15 kB) to quickly complete a considerable number of short TCP transfers [19].

Experiments by Google [20] showed benefits in reducing web object transfer times at moderate cost in terms of increased congestion and associated packet losses when used over their network. This motivated an experimental update to TCP in 2013 [19].

This specification [19] also recommended TCP implementations to refrain from resetting IW to one segment unless multiple SYN or SYN-ACK retransmissions occurred or congestion losses confirmed. The current standard specifies resetting IW to 1 even when a single control packet is missing. However, considering RFC 6298 reduction of initial RTO from 3 s to 1 s, this would excessively penalise connections with high RTTs (e.g. satellite links).

Current analysis of using a large initial window has not explored the potential collateral damage to other flows that share a bottleneck where the large IW is continuously used.

An adaptive approach to set IW was proposed in [21]. Instead of defining a particular value for IW or prescribing a schedule for increase over time. This proposed an automatic mechanism to increase the IW by objectively measuring when an IW was too large, allowing IW to be adjusted automatically. Generic methods to choose a suitable IW remain an item of future research.

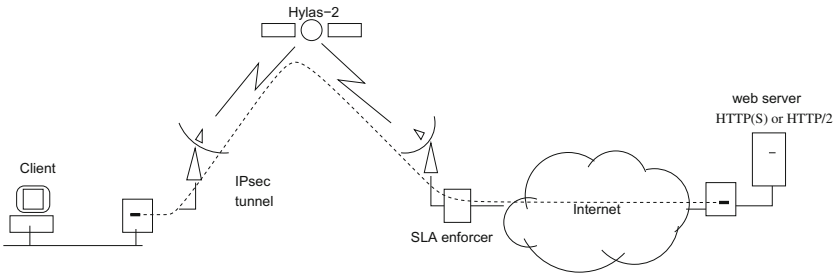

Fig. 1. HTTP/2 experimental testbed topology based on Hylas-II satellite platform

4 Performance Evaluation of HTTP over Satellite

A range of experiments evaluated the page load time (PLT) using both HTTP/1.1 and HTTP/2, and the impact of the different TCP stack modifications. Our tests (Fig. 1) accessed a web server via satellite terminals using the Hylas-II satellite network operated by Avanti^{TM} [22] based on IPoS [23]. Metrics other than the PLT have been used to evaluate web performance. These include the time-to-the-first-paint and the object OnLoad time [10]. However, we preferred to use the PLT which reflects both network performance and user experience.

Avanti^{TM}provided various SLA options for Return Link (RL) and Forward Link (FL) bandwidth provisioning. A traffic shaper co-located with the Satellite Gateway acted as SLA enforcer for the FL. Peak rates in our settings were 512 kb/s for the RL and 2048 kb/s for the FL. The satellite network implemented a range of transport and application layer PEP techniques. Application-layer PEP mechanisms were disabled when using HTTP/2 due to the use of SSL the encryption of the TCP payload.

By default, a transport PEP splits the TCP connection into three parts, one between client and terminal, one between terminal and satellite gateway, and one between gateway and server. To evaluate the impact of Transport PEPs, some tests disabled PEP by using IPsec encryption to hide the TCP header. This allowed us to study a HTTP session with a direct connection between server and client.

We also performed experiments using a network emulator introducing an artificial RTT of 750 ms, which approximated the Internet plus observed satellite round-trip. This was used as a reference case, since it avoided the varying delay from the dynamic bandwidth allocation used by the satellite system.

The web server ran on a Linux-based platform with kernel 3.8. This kernel implemented the recent TCP modifications for fast start-up. In particular, the kernel used RFC6928 by default. All experiments used the Mozilla Firefox (vers. 31) web browser, which was equipped with a protocol analyser to capture the HTTP/2 conversation between the client and server. The Firefox configura-

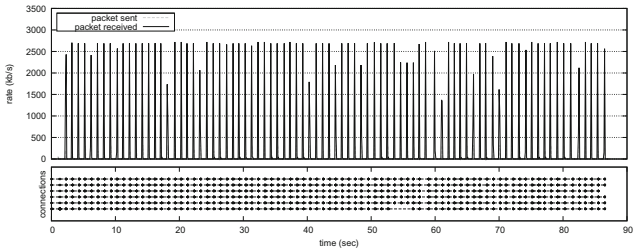

Fig. 2. Dynamics of HTTP traffic rate

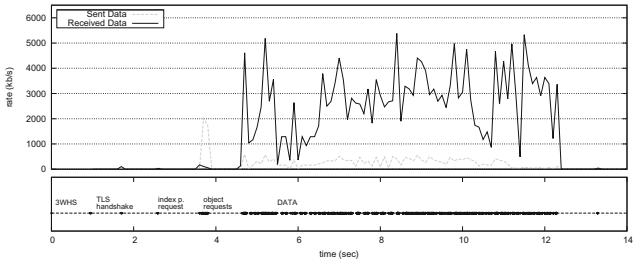

Fig. 3. Dynamics of HTTP/2 traffic rate

tion panel allowed the user to switch between HTTP and HTTP/2. The implementation of HTTP/2 was based on the SPDY/3 module provided by Google™.

To explore a range of web page compositions, we considered three cases of web-pages lengths (of 500 kB, 1500 kB, and 2500 kB respectively) and three cases of object sizes (of about 5 kB, 20 kB and 100 kB respectively). This resulted in 9 combinations of pages with homogeneous object size of different length. The number of objects in our tests varied between 5 (when the page size is 500 kB and the object size is 100 kB) and 500 (when the page size is 2500 kB and object size is 5 kB). Although pages with more than 200 objects are unlikely in today's Internet, the number of objects per page has steadily increased in recent years and HTTP/2 should be devised having in mind the extreme cases. This approach was also followed in another HTTP/2 study [10]. Each experiment was repeated 10 times to mitigate the variability of download duration on the average PLT.

4.1 Rate Patterns in HTTP and HTTP/2

Figure 2 illustrates the transport dynamics for HTTP/1 with a web-page of 500 objects of around 5 kB[1]. The figure also plots the activity, i.e., when packets are received/transmitted, of each of the six parallel HTTP/1 client-server connections that Mozilla Firefox opens towards the server. Each connection carries around one-sixth of the total number of objects and each object is downloaded by a separate request/response transaction.

[1] Rate samples were taken every 100 ms.

Performance over satellite were influenced by the *connection retry timeout* in the Firefox configuration which is set by default at 250 ms. This parameter sets the time Firefox has to wait after opening a connection before trying to open a new connection. A too small value for the connection retry timeout caused multiple connections being opened and connection errors at start.

Since the objects are small and each connection can carry at most one object per RTT, six connections are far too few to utilise the large bandwidth-delay product path (about 650 kB in this case). This results in a low throughput (230 kb/s) which is much less than the 8 Mb/s available bandwidth. Moreover, HTTP performance cannot be improved by transport PEPs because only a few kilobytes are delivered at each request/response over a TCP connection. The amount of data per transaction is therefore smaller than the TCP IW and could be sent end-to-end by TCP in one RTT.

A very different output is observed instead when HTTP/2 is used (Fig. 3). Indeed, the same 500 HTTP objects can be delivered in about 12 s with an average throughput of 1.7 Mb/s using HTTP/2. The better result is obtained mainly due to object multiplexing. Once that the HTTP/2 client has received the index page, it can generate a sequence of object requests issuing stream synchronisation messages (SYN_STREAM). Each stream is identified by a *stream-ID* which is used by the server to refer to a particular object in the response. The transmission of SYN_STREAMS is shown at around time 4 s when a 2 Mb/s peak rate by the browser. As multiple request messages can be sent simultaneously, the server can receive request messages faster, respond to many more request at the same time and maintain a high throughput. The throughput is however not optimal due to the limited number of concurrent streams allowed by the web server: The default HTTP/2 configuration in Apache allows only up to 100 concurrent streams per session, while the path could sustain up to 150 streams.

A significant amount of time (about 3 s) is spent in connection opening. This includes the TCP three-way handshake (one RTT) and a full TLS handshake (two RTTs). Both TCP and TLS connection establishment require the synchronisation between server and client and hence full satellite RTTs to complete. However, a three RTT connection opening is strictly required only the first time the site is accessed. In a subsequent access the web client could indicate the session-ID of a previous TLS session to resume the session (if session caching, RFC 5246, is supported by the server) or use a session ticket, RFC 5077, which was previously released by the server for a certain TLS session. Both these mechanisms reduce the TLS handshake to one RTT (the *abbreviated* TLS handshake). In addition, the client could use a TCP Fast Open cookie to send data on SYN and eliminate the TCP handshake entirely.

4.2 Impact of Page Composition on Web Performance

Figure 4 illustrates the PLT of test web-pages with respect to the number of objects for HTTP and HTTPS when the pages were accessed through the PEP. The results of our experiments suggest that the number of objects is an important parameter in determining the PLT with HTTP/1 and HTTPS. These results

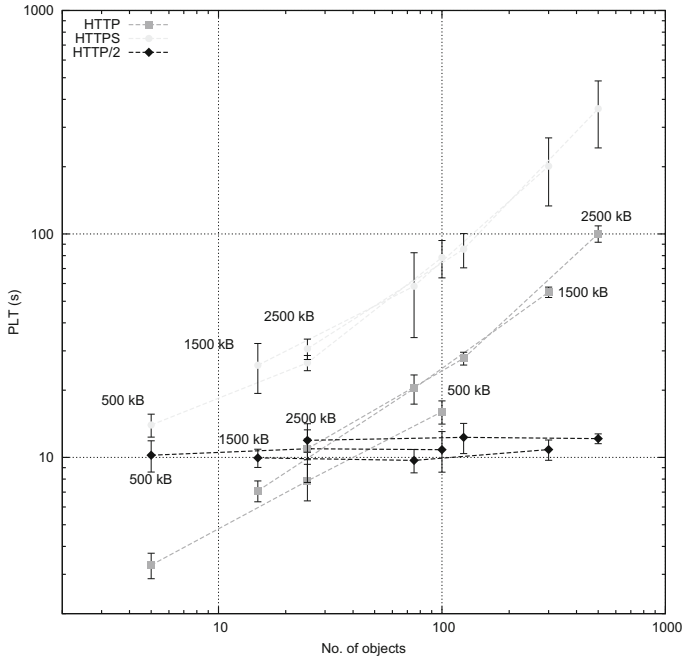

Fig. 4. Page load time (PLT) with respect to the number of objects for HTTP, HTTPS and HTTP/2 with satellite split connection. The object size is the ratio between the page size (label) and the no. of objects.

were obtained with a FL capacity of 8 Mb/s and the default Linux configuration for the TCP/IP stack.

Although the PLT exhibits large variability, an increasing trend of PLT with respect to the number of object is evident in all cases. The number of objects influences the PLT more than the page length. For example, downloading a web page consisting of a hundred objects of 5 kB (the *500 kB* label in the picture) takes around 35 s when using HTTPS with direct access, while a page of 15 100 kB objects (*1500 kB* label) takes only 12 s.

The dependency of PLT on the number of objects with HTTP is not surprising. When a web page is accessed using HTTP/1 or HTTPS, the web browser opens a certain number of connections towards the web server to request and retrieve web objects. Since only one object can be requested on a connection and only after the transmission of a previous object is completed, the throughput of a web session (T_{http}) is limited to transferring one average object size (S_{obj}) per round-trip time (RTT) per connection, i.e., $T_{http} \approx N_c \times S_{obj}/RTT$. This rate limitation imposed by the HTTP/1 request/response model is particularly a limitation for long delay paths, and is one of the main motivations for using application-layer PEP.

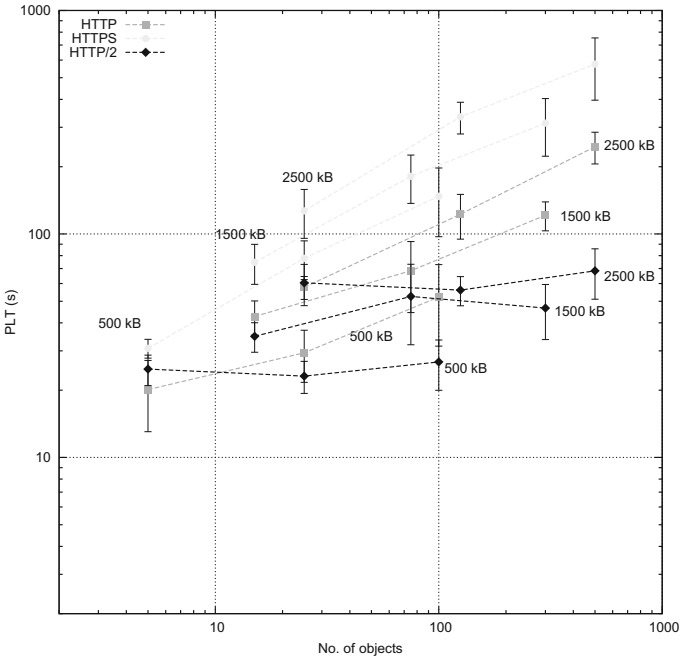

Fig. 5. Page load time (PLT) with respect to the number of objects for HTTP(S) and HTTP/2. The object size is ratio between the page size (label) and the no. of objects.

Figure 4 also shows that the PLT with HTTP/1 is significantly lower than with HTTPS when a page is accessed directly. In reality, this performance is only obtained because the HTTP/1 connection is not persistent and represents the worst case for HTTP/1. If the connection is not persistent, a new TCP and TLS handshake needs to be performed for each object. However, when persistence is enabled, the performance of HTTP/1 and HTTPS are similar.

When HTTP/2 is used (Fig. 4), the PLT is around 10–12 seconds, and is not strongly dependent on the number of objects. This again is not surprising considering that many more objects can be transmitted concurrently. However, we notice that HTTP/2 performance depends weakly also from the page size. This is probably due to the high variability of the rate of the TCP session (as illustrated in Fig. 3). As the throughput can vary substantially from one experiment to the next, the impact of the page size on the PLT is somehow overshadowed by other factors, including the interaction between TCP congestion window (cwnd) and the HTTP/2 request/response mechanism.

4.3 Performance with Large End-to-end Delay (no Transport PEPs)

Figure 5 shows the performance when a client connects to the server without intermediaries. Although the plot exhibits more variability than the one with

PEPs, we can draw similar conclusions. Inspecting the traces, we found that RTT samples are affected by very large variations (on the order of seconds), most likely due to queuing delays at the FL access queue. While these delays are mitigated by the splitting of the connection in the PEP case, using the IPsec tunnel these delays affect the end-to-end RTT. This reflects poorly on the TCP RTO estimation and ultimately on the performance.

The average PLT with HTTP/2 ranges between 11 and 15 s.The PLT with HTTPS spans a much larger interval, taking up to few minutes to download the page when the number of objects is more than one hundred. The performance improvement of HTTP/2 is clearly related to the ability of the protocol to efficiently multiplex many small objects onto the same TCP connection. However, the throughput of HTTP/2 was less than the case with split connection and much less than the available bandwidth (8 Mb/s). This suggests that capacity is not the primary limiting factor on the HTTP/2 performance. Analysing the traces we found that several other factors contributed to reduce the throughput with respect to the split-connection case:

1. The maximum number of concurrent active streams was limited to one hundred (default value in Apache server). This is done as a precautionary measure to constrain the amount of memory used by HTTP/2 streams. While this number is sufficient for a terrestrial wired session, satellite networks could easily host several hundreds of streams if their size is limited to few kilobytes.
2. The memory reserved by the client for the TCP connection receive buffer was about 128 kB. Although TCP Window Scaling Option [24] was used (as in the majority of modern TCP/IP stack) and the application was able to retrieve data quickly from the receive buffer, the TCP receiver was forced to advertise a window that was smaller than the bandwidth-delay product. The small advertised window was due to the limited buffer space reserved by the application.
3. The browser fired a timeout to reset the connection when it was idle for few seconds. In some of our tests, the HTTP session was reset even more than once to finish downloading the objects. At each reset new TCP connections were created.

All these effects are clearly related to the fact that the HTTP session was optimised for a terrestrial Internet path. However, HTTP/2 offers mechanisms (such as the SETTINGS and UPDATE control frame) to configure the HTTP session to the actual client/server requirements. Using these mechanisms it is possible to optimise the session by avoiding parameters statically configured in the browser.

5 Conclusion

This paper reports the results from a series of tests using SDPY/3 (the most recent HTTP/2 implementation from Google^{TM}) over the satellite platform. Our results clearly show that when the number of multiplexed objects is large,

HTTP/2 largely outperforms HTTP(S). In particular, we observed that when the web page is made up of more than hundred objects HTTP/2 completes the web-page transfer in around ten seconds while the HTTP(S) spends several tens of seconds.

On the other hand, with less than ten objects HTTP performs better than HTTP/2. This is because HTTP does not need to wait for the TLS handshake to complete and benefits of the presence of application-layer PEPs. However, HTTP/2 performance could have been improved if protocol parameters had been configured for the satellite delay. For example, the default size of TCP send and receive buffer of the HTTP/2 connection was around 130 kB. While this did not have particular effects when the end-to-end connection was split by the intercepting PEP, it proved insufficient when the client and server were connected directly without the mediation of PEPs. Also, the default initial connection size used in the Flow Control mechanism and the maximum number of permitted parallel streams were inferior to the ones allowed by the bandwidth-delay product. Finally, the HTTP/2 *connection-retry-timeout* was set to 250 ms, which is unrealistic for the satellite case.

PEPs are nowadays essential to achieve satisfactory web performance over satellite. In particular, split connections are required to avoid the long delays created by TCP end-to-end congestion control. However, they have many drawbacks. PEPs break TCP end-to-end semantics leading in some cases to drop TCP options or ignore new options (for instance in our tests split-TCP prevented the use of large window option requested by the client). This somehow limits the ability to upgrade the system when new Internet transport techniques are made available, thus contributing to the *ossification* of the Internet, i.e. the lack of progress due to compatibility issues. Also, PEPs make the system more complex to configure and maintain since transport code need to be run at the terminals and gateway of the satellite network. In other words, PEPs are a *necessary evil*, not loved by network operators.

HTTP/2 may be seen as a way forward to address the performance issue of web traffic over satellite. HTTP/2 offers the ability to send an entire web page over a single TCP connection by introducing methods to multiplex request/response transactions (or more in general web streams) within a persistent connection. HTTP/2 also introduces a mechanism to *push* data to the client without an explicit request, thus saving precious RTTs. The HTTP/2 improvements are complemented at the TCP layer by a series of proposals, such as TCP Fast Open and the larger Initial Window, to improve TCP responsiveness at startup. These recommendations highlight inefficiency of the current network stack and remove inessential RTTs. Combined with the new application-layer protocol, these modifications are expected to speed-up significantly web sessions.

Our tests have shown that HTTP/2 is probably not yet mature technology to definitely replace the traditional satellite web architecture based on PEPs. However, the problem we identified are implementation-related and can probably be overcome by revising HTTP/2 specifications having in mind the issues introduced by the large RTT.

References

1. Domenech, J., Pont, A., Sahuquillo, J., Gil, J.: A user-focussed evaluation of prefetching algorithms. ACM Comp. Comm. **30**, 2213–2224 (2007)
2. Ramachandran, S.: Web Metrics: Size and number of resources (2012)
3. Berners-Lee, T., Fielding, R., Frystyk, H.: Hypertext Transfer Protocol - HTTP/1.0. Internet-Draft draft-ietf-http-v10-spec-05, IETF Secretariat (1996)
4. Fielding, R.T., et al.: Hypertext Transfer Protocol - HTTP/1.1. RFC 2616, RFC Editor (1999)
5. Allman, M., Paxson, V., Blanton, E.: TCP Congestion Control. RFC 5681. RFC Editor (2009)
6. Elkhatib, Y., Tyson, G., Welzl, M.: Can SPDY really make the web faster? In: proceedings of IFIP Networking Conference, Tronhdeim (2014)
7. Peon, R., Belshe, M.: SPDY protocol - Draft 3 (2012)
8. Roskind, J.: QUIC (QUIC UDP Internet Connections), Multiplexed Stream Transport over UDP Google Technical report (2012)
9. Belshe, M., Peon, R., Thomson, M.: Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, RFC Editor (2015)
10. Wang, Z.S., Balasubramanian, A., Krishnamurthy, A., Wetherall, D.: How Speedy is SPDY? In: 11th USENIX NSDI, Seattle, pp. 287–393 (2014)
11. Caviglione, L., Gotta, A.: SPDY over high latency satellite channels. EAI Endorsed Trans. Mobile Comm. Appl. **2**, 1–10 (2014)
12. Luglio, M., Roseti, C., Zampognaro, F.: SPDY multiplexing approach on long-latency links. In: Proceedings of IEEE WCNC, Instanbul, pp. 3450–3455 (2014)
13. Luglio, M., Roseti, C., Zampognaro, F.: Resource optimization over DVB-RCS satellite links through the use of SPDY. In: Proceedingsof WiOpt, Hammamet (2014)
14. Radhakrishnan, S., Cheng, Y., Chu, J., Jain, A., Raghavan, B.: TCP Fast Open. In: Proceedings of ACM Conext 20133, Tokyo (2011)
15. WalkerSand: Quarterly Web Traffic Report. Technical report, WalkerSands Communications, Chicago (2013)
16. Cheng, Y., Chu, J., Radhakrishnan, S., Jain, A.: TCP Fast Open. RFC 7413, RFC Editor (2014)
17. Braden, B.: T/TCP - TCP Extensions for Transactions Functional Specification. RFC 1644, RFC Editor (1994)
18. O'Hara, B., Calhoun, P., Kempf, J.: Configuration and Provisioning for Wireless Access Points (CAPWAP) Problem Statement. RFC 3990, RFC Editor (2005)
19. Chu, J., Dukkipati, N., Cheng, Y., Mathis, M.: Increasing TCP's Initial Window. RFC 6928, RFC Editor (2013)
20. Dukkipati, N., et al.: An argument for increasing TCP's initial congestion window. ACM SIGCOMM Comp. Comm. Rev. **40**, 26–33 (2010)
21. Touch, J.: Automating the Initial Window in TCP. Internet-Draft draft-touch-tcpm-automatic-iw-03.txt, IETF Secretariat (2012)
22. Avanti: Avanti Satellite Communications (2014)
23. Hughes Networks: IP over Satellite (IPoS) - The Standard for Broadband over Satellite (2007)
24. Jacobson, V., Braden, B., Borman, D.: TCP Extensions for High Performance. RFC 1323, RFC Editor (1992)