Performance Analysis of Next Generation Web Access via Satellite

R. Secchi[1], A. C. Mohideen[1], and G. Fairhurst[1]

[1]School of Engineering, University of Aberdeen, , Fraser Noble Building, AB24 3UE, Aberdeen (UK)

Abstract

Responsiveness is a critical metric for web performance. Recent work in the Internet Engineering Task Force (IETF) and World Wide Web Consortium (W3C) has resulted in a new set of web protocols, including definition of the hypertext transfer protocol version 2 (HTTP/2) and a corresponding set of TCP updates. Together, these have been designed to reduce the web page download latency compared to HTTP/1.1. This paper describes the main features of the new protocols and discusses the impact of path delay on their expected performance. It then presents a set of tests to evaluate whether current implementations of the new protocols can offer benefit with an operational satellite access network, and suggests how the specifications can evolve and be tuned to further enhance performance for long network paths.

1 Introduction

The Hypertext Transport Protocol (HTTP) was created in the late nineties as a method to access and navigate a set of linked web pages across the Internet [1, 2]. HTTP/1.x used a request/response model, where each web object was requested separately from the other objects. Since development, limitations of this protocol have become apparent [14].

The original protocol did not scale to support modern complex web pages. Protocol chatter (i.e., protocol exchanges that need to be synchronised sequentially between end-points) within HTTP/1.1 added significant latency [5], as also did the protocol interactions with security and transport layers (i.e., Transmission Control Protocol [4]). This caused significant overhead at start-up, especially when used over a long network path, such as a satellite broadband access network [3].

Early web pages typically consisted of a few tens of kilobytes of data and were normally static (where content was not constructed dynamically by the server). The last ten years have seen a radical change in the content of typical web pages, with pages becoming more complex, and often consisting of (many) tens of elements, including images, style sheets, programming scripts, audio/video clips, HTML frames, etc. [6, 7] Moreover, many forms of content retrieved by web requests are updated in real-time (e.g., linked to live events or dynamic databases). Applications for HTTP now extend beyond browsing applications, e.g., to interact with distributed web applications, support for conferencing and a wide range of other uses.

The growing size and number of objects per page led to the common practice of web clients using many concurrent HTTP/1.1 connections, and web content providers using "sharding" to distribute their content across multiple server sites and "spriting" of images. These techniques were able to decrease the down-load time. But, while multiple TCP connections can reduce page load times, this is not a network-friendly solution because it can also result in traffic competing aggressively with other network flows, e.g., disrupting time-critical streaming applications (see description of the impact of collateral damage in [5]).

The design of HTTP/1.1 is also vulnerable to head-of-line blocking (HoLB), because it sequentially serves requests and responses. When an important object is delayed by a less important object, this adds latency to the initial drawing of a web page at a client (also known as *first paint*) [5, 35].

To improve responsiveness of page download, Google™ proposed an alternative to HTTP/1.1, which became known as SPDY [8]. In 2009, the initial results from using SPDY triggered formation of the HTTPbis working group in the Internet Engineering Task Force (IETF) and related work in World Wide Web Consortium (W3C). Together this led to the definition of a new standard, HTTP/2 [9] in 2015. The adoption of HTTP/2 has been growing rapidly [10] and is expected to continue.

Work on HTTP/2 has been accompanied by developments of the associated protocols. At the transport layer, this has motivated updates to the TCP specifications [36] and a proposal for an alternative delivery protocol, QUIC [11, 12, 13].

[14] provided an extensive comparison of HTTP(S) and HTTP/2 for common network path characteristics. This confirmed that HTTP/2 effectively addressed many of the problems of HTTP, but also highlighted some potential pitfalls. It observes that a single connection is more exposed to transient packet loss than multiple HTTP/1.1 connections (significant for high delay paths), and that the performance of HTTP/2 could be less when used in the current highly *sharded* web. The benefit can also be limited by current page design [15]. The design of new content may be expected to evolve to favour HTTP/2, as this becomes more widely deployed.

This new web architecture is designed as a replacement for HTTP/1.x, and must therefore be able to operate across all network paths encountered in the Internet. It is important to consider not just the performance for high speed broadband services, but also network paths that have very different characteristics. In this context, it is useful to evaluate the performance over a long path, such as offered by a broadband satellite service.

Services using geostationary satellites incur a higher than average path delay (typically 500-750 ms), due to a combination of physical-layer propagation delay and the time required by sophisticated medium access control (MAC) protocols. This higher path delay can expose inefficiencies in protocol design/usage - and as for other services with a higher path delay has led to common use of protocol accelerators (sometimes known as Protocol Enhancing Proxies, PEPs [19]) with HTTP/1.1. This is therefore a helpful test case for HTTP/2 and helps answer the question as to whether the performance enhancements in the new protocol are sufficient to eliminate the need for PEPs in satellite networks or whether new methods will be needed.

This paper complements the analysis on HTTP/2 over satellite done in [16, 17, 18, 3] by evaluating the protocol across a wider range of scenarios, including cases with PEPs and cases where PEPs were unable to offer benefit (e.g., with encrypted tunnels). Our results confirm previous analysis that HTTP/2 offers better performance with respect to HTTP/1.1 provided that the system is fine-tuned to support the protocol. In [16], for example, it was observed that multiplexing the entire web connection over a single connection can be a disadvantage when the connection suffers wireless loss. However, this performance reduction can be compensated for by using an appropriate scheduling discipline and countermeasures for the wireless errors. [18] showed that the performance of HTTP/2 is sensitive to the bandwidth allocation method used by a satellite system. However, a proper choice of the Bandwidth on Demand (BoD) leads to a reduced Page Load Time (PLT).

The remainder of this paper is organised in a series of sections. Section 2 introduces the new features of HTTP/2 and discusses their use over high delay paths. Section 3 surveys proposals for TCP modifications and their impact on web performance. Section 4 reports the results of a set of experiments using a satellite service, followed by a discussion in Section 5. Section 6 provides conclusions.

2 HTTP/2

Although HTTP/2 was designed to replace HTTP/1.x, deployment is envisaged where HTTP/2 will co-exist with older HTTP/1.x systems for the foreseeable future, since not all servers will immediately change to use HTTP/2. The design of HTTP/2 therefore chose to describe web resources using the URI scheme developed for HTTP/1.x and HTTPS, although it makes also available the "h2" and "h2c" URI schemes (for native HTTP/2 over TLS and HTTP/2 cleartext, respectively). To facilitate use of existing middleboxes (e.g., PEPs, firewalls and network and port translators, NAPTs) it also reuses the same TCP port numbers (80 and 443). HTTP/2 sessions can be unencrypted or encrypted using the Transport Layer Security (TLS) protocol. Use of TLS is default, as is the use of end-to-end compression of web data. In addition, HTTP/2 defines a compressed HTTP header (HPACK [20]) eliminating redundant HTTP/1.1 headers.

2.1 Connection Setup

A key goal of HTTP/2 is to remove the latency introduced by protocol chatter - a term we use to describe unnecessary protocol interactions across the network path. Each interaction typically incurs a round trip time of latency, increasing the PLT, especially significant when the path delay is large.

A method was required to signal the use of HTTP/2. HTTP/1.1 defined the *Switching Protocol* header that, when inserted in a request, indicates the clients intent to change the session protocol. This mechanism costs an extra RTT after connection establishment.

Instead, HTTP/2 performs an Application Layer Protocol Negotiation (ALPN) extension (sent to signal the presence of HTTP/2 during the TLS handshake).

HTTP/1.0 [1] allowed only one web object to be requested at a time over a TCP connection. HTTP/1.1 [2] permitted use of persistent HTTP connections, i.e., allowing reuse of a TCP connection for subsequent object requests. This could avoid the overhead of transport connection establishment - useful for long delay paths [35], although it was not widely used. HTTP/2 has adopted this persistent design, where the bundle of HTTP request/response transactions required to retrieve a web page are mapped to a single transport connection.

2.2 Multiplexing of Frames and Streams

In HTTP/2, web page resources (objects) are sent in *Frames*. Each stream comprises independent, bi-directional sequences of Frames exchanged between a server and the client. Streams can be initiated by either a client or a server. (In HTTP/1.1, an HTTP connection could only be started by the client). An HTTP/2 connection can have multiple concurrently open streams, with either end-point interleaving frames from multiple streams. This removes the need to open multiple transport connections to minimise HoLB. Specific frames may be prioritised to minimise its HoLB.

Once an HTTP/2 client has received the index page, it may generate a sequence of requests using stream synchronisation messages (SYN_STREAM). Each stream is identified by a *stream-ID*. The server uses this to refer to the objects in the responses.

Multiplexing objects can improve reduce the PLT for a long network path [3]. This reduces both the chatter from opening new connections and reduces the volume of traffic by sending more data per transport segment [5]. By eliminating the advantage of parallel connections and the performance incentive of server sharing/spriting, this also encourages use of fewer transport connections. This in turn reduces the number of simultaneously active ports, simplifying the design of PEPs and NAPTs and may benefit other network functions (e.g., easier identification of traffic expectations by an adaptive satellite MAC).

2.3 Flow Control

HTTP/2 introduces a credit-based Flow Control mechanism, essential for a multiplexed design that removes HoLB. This seeks to regulate the connection rate to protect resource-constrained endpoints (or intermediaries that process HTTP Streams) from being overwhelmed by excessive received data. This helps in cases where the receiver/intermediary is unable to process data on one Stream, yet wants to continue to process other Streams. In a multi-hop HTTP/2 connection each receiver can control the rate over its local path.

The HTTP/2 specification defines the necessary tools to implement Flow Control, but it does not mandate a particular method of use. Clients specify an initial size for each stream in the SETTINGS frame, indicating the amount of data they are prepared to accept for the entire connection. During the session, clients may use WINDOW_UPDATE frames to increase the size of a stream or the connection. The WINDOW_UPDATE frame specifies the increment in the size of the Stream.

Briscoe [21] noted in a memo to the HTTP/2 working group that the credit-based HTTP/2 flow control mechanism lacks a technique to detect congestion at the application-layer and is not capable of adapting the HTTP/2 transmission rate for a path with a large bandwidth delay product. A receiver could restrict its rate because it lacks feedback information needed to increase the size of the flow. Too small an initial setting may constrain the transfer rate over a path with a high bandwidth-delay product connection. Section 4 investigates this issue in a current browser.

2.4 Server Push

HTTP/2 defines an experimental Server Push mechanism. This allows a server to initiate a Stream to *push* additional responses to a client. This could be useful when a server can anticipate resources (objects) that a client may need (e.g., based on previous requests).

Server Push can be 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 containing the pushed data. If the client decides to accept the pushed data, it does not need to send a request. Otherwise, the client can issue a control Frame (RST_STREAM) that stops transmission of the pushed data.

Server Push could be used as performance tool for long paths, where it could potentially reduce the PLT by at least one RTT. For example, eliminating the need to request related objects linked to an index page. However, predicting the best data to push can be difficult. In some contexts, the unsolicited data may be undesirable, for instance when it transfers objects that are either not desired by a client, duplicates data already present in a receiver

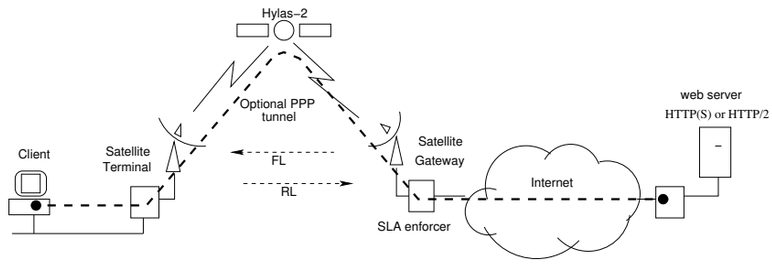


Figure 1: Topology of the HTTP/2 experimental long network delay testbed using the Hylas-II satellite platform

cache [15], or when it consumes additional network capacity that is charged or could otherwise be used for more important data. To avoid *pushing* useless data, a client can request that Server Push may be disabled when the connection settings are negotiated. The benefits for a mobile data or satellite network are therefore difficult to predict.

3 Transport enhancements that can improve web performance

Transport protocol mechanisms (such as connection-setup, congestion control and loss recovery) also contribute to the PLT. The benefit of introducing new HTTP mechanisms would be limited if these were not complemented by changes at the transport layer. Awareness of this dependency has promoted a series of proposals to update TCP [4] since 2010. The underlying mechanisms are also applicable to the Quick UDP Internet Connection (QUIC) [11, 12], an experimental UDP-based protocol that seeks to combine a set of mechanisms functionally equivalent to a combination of TCP, TLS, and HTTP/2. QUIC has seen experimental deployment by Google, using congestion control methods similar to those of TCP [11, 13].

This section surveys recent relevant transport layer enhancements. Although most enhancements have been motivated by a desire to improve performance over terrestrial networks, they can improve performance for long network paths. Removing even a few initial RTTs can have macroscopic benefits for a flow using a long path.

3.1 Fast Open

TCP uses an initial three-way handshake to setup each connection; this introduces one RTT of latency before data can be sent. Although it has long been possible to exchange one idempotent segment of data in the initial SYN message using T/TCP [22], this had drawbacks and saw limited use.

The TCP Fast Open (TFO), [23] experimental specification, published in 2014, addressed the pitfalls of T/TCP by introducing a cookie mechanism [36] to authenticate clients with a server when they first start a connection. This benefits an HTTP client by saving setup latency when a client needs to connect multiple times to the same server over a path with a high network delay [5]. This is particularly beneficial for high delay paths carrying requests made over non-persistent connections or when content has been sharded, but can offer benefit in other practical situations.

3.2 Larger Initial Window

After opening a TCP connection, the congestion window grows each RTT from the Initial Window (IW) value [24]. This limits the transfer rate of transport connections that transfer small amounts of data. An IW of 3 segments was widely deployed to improve TCP Fast Retransmit, but was insufficient to carry many small web objects. Experiments by Google [25] showed benefits in reducing web object transfer times at moderate cost in terms of increased congestion and associated packet loss when used over their network. This motivated an experimental update to TCP in 2013 [26] to increase the server IW to 10 segments (IW10 corresponding to about 15 KB).

[26] also recommended TCP implementations refrain from resetting IW to one segment unless multiple SYN or SYN-ACK retransmissions occurred or congestion losses were confirmed. However, the current standard specifies resetting IW to 1 even when a single control packet is missing. Also, considering that some operating systems (e.g., Linux) use a non-standard initial Retransmission TimeOut (RTO) lower than one second, the IW reduction can penalise connections with a high network path delay.

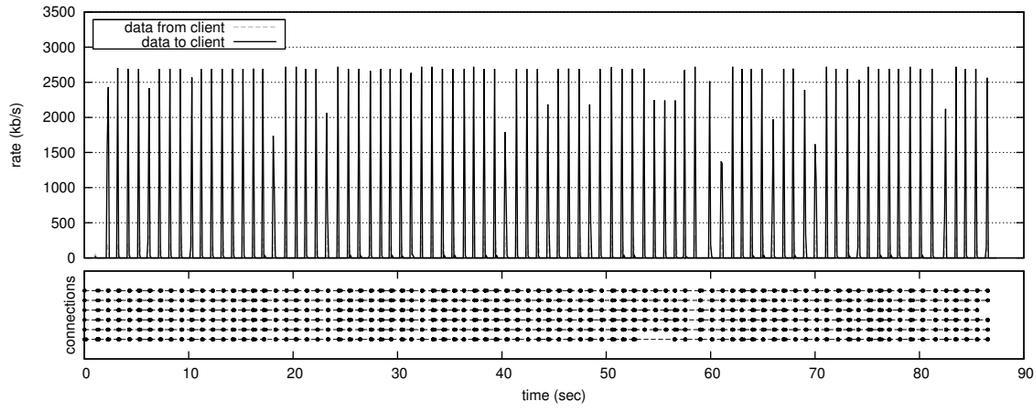


Figure 2: Dynamics of HTTPS traffic rate and active connections (shown below). Rate samples were taken every 100 ms.

4 Performance evaluation of HTTP over satellite

A set of tests explored the impact of a long network path using experiments to assess the performance of an HTTP/1.1 and an HTTP/2 server. The experiments used the Apache 2.2 web server running under Linux kernel 3.8. This Linux kernel implements the TCP enhancements described in section 3 (IW 10). The HTTP/2 implementation was based on the SPDY/3 module (*mod-spdy*) provided by Google™. All experiments used the Google Chrome™(vers. 36) web browser client and captured protocol messages using the built-in protocol analyser. The Google Chrome configuration panel allowed a switch between HTTP and HTTP/2.

The experiments used an IPoS [27] terminal with the Hylas-II satellite operated by Avanti™ [28], shown in Figure 1. The satellite ISP set the peak rates for the Forward Link (FL) as 8 Mb/s and the Return Link (RL) as 512 KB/s according to a commercial Service Level Agreement (SLA), (reflecting current standard industry practice). The traffic was managed by a SLA enforcer co-located with the satellite gateway. The satellite network employed a transport layer PEP [19]. This split a TCP session, breaking the end-to-end transport connection to the HTTP server at the satellite terminal and/or satellite gateway. In the experiments, this PEP function could be disabled by using a tunnel over the satellite service [3].

Our methodology is similar to another HTTP/2 study [15] which measures the PLT to characterise network performance and user experience. This is the time to retrieve all the data associated with a web page, while not taking into account the time to render the web-page on screen. This metric was chosen for ease of measurement, rather than the time-to-first-paint [18], but was still able to evaluate the performance benefits of an HTTP protocol. Each experiment was repeated 10 times to mitigate the variability of the download duration on the average PLT. The Figures report the 95% confidence interval together with the average PLT.

To explore a range of web page compositions, three sets of web-pages were considered (500 KB, 1500 KB, and 2500 KB) with three sets of object sizes (respectively of about 5 B, 20 KB and 100 KB). This resulted in 9 combinations of pages each with a different size, but objects of a similar size.

The number of objects in our tests was varied between 5 (for a page size of 500 KB and an object size of 100 KB) and 500 (for a page size of 2500 KB and object size of 5 KB). HTTP/2 has been designed to accommodate large numbers of objects per page, and hence is insensitive to how web data is divided into objects. The number of objects per page has steadily increased in recent years, although pages with more than 200 objects are not normally found in today's Internet.

4.1 Observed download dynamics for HTTP(S) and HTTP/2

Figure 2 and 3 illustrate the performance benefits of HTTP/2 with respect to HTTPS. Both diagrams show the rate of an HTTP web session with a web-page of 500 objects (each approximately 5 KB) measured using rate samples every 100 ms. The plot in the lower part of the figure shows the activity, i.e., when packets are received/transmitted over each connection. All connections were persistent between the client and the PEP while the PEP created a new TCP connection for each HTTP request/response transaction.

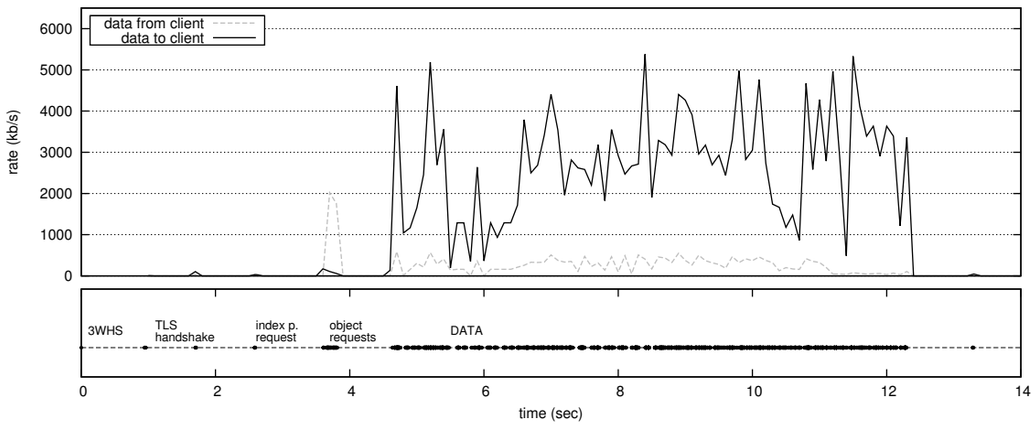


Figure 3: Dynamics of HTTP/2 traffic rate and state for a single connection (shown below). Transfer rate for a web session over the satellite forward link (solid line) and RL (grey line). Rate samples were taken every 100 ms.

When using HTTP/1.1 (Figure 2), the client opened six parallel transport connections each carrying a single request/response transaction. The server sent objects of a few kilobytes with a size smaller than the TCP IW, and hence each object was sent in one RTT. Since each connection could carry only one object per RTT, the six connections were too few to be able to utilise the bandwidth-delay product (about 650 KB in this case) of the satellite. This limited the transfer rate to 230 KB/s, much less than permitted by the SLA (8 Mb/s). This HTTP performance could not be improved by a transport PEP, because the transmission time was determined by the end-to-end time to complete each response.

On the other hand, HTTP/2 (Figure 3) allowed the same set of objects to be delivered in about 12 s with an average transfer rate of 1.7 Mb/s. This performance was mainly due to HTTP/2 being able to request simultaneous Streams using a single transport connection (the connection activity is shown in the lower part of Figure 3). The transmission of SYN_STREAMS at time 4 s achieved a peak download rate of 6 Mb/s. This is still less than the rate permitted by the SLA, but is a significant increase with respect to HTTP/1.1. Multiple simultaneous request messages allowed the server to receive requests faster and to respond to several requests at the same time, achieving a higher transfer rate.

4.2 Improving further HTTP/2 performance

Figure 3 shows also that a significant time of HTTP/2 PLT is spent in opening the connection (one RTT for the TCP three-way handshake) and in TLS handshake (two RTTs).

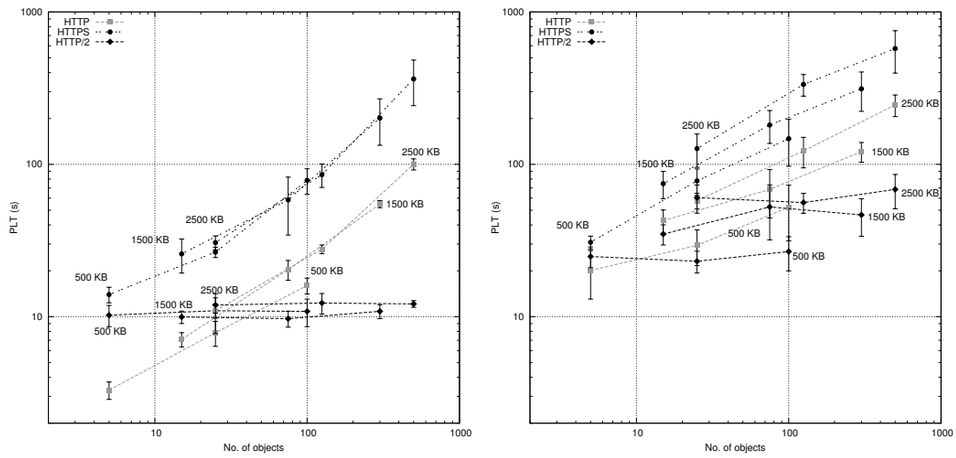
However, three RTTs are required only when a server is first accessed. A TFO-capable client can entirely eliminate the TCP handshake when communicating with a TFO-enabled server. A client can also reduce the cost of the TLS handshake in HTTP/2 to one RTT (the *abbreviated* TLS handshake) by using one of two methods: It could resume with an indication of a previous session-ID to a server that supports Session Caching [29], or could use a Session Ticket [30] that was previously released by the same server.

The cost of a first-time TLS handshake to a server can also be reduced to one RTT using the TLS False Start procedure [31]. This allows a client to send data as soon as a shared key is calculated, normally at the end of the first RTT after the client has verified the identity of the server. After sending the shared secret to the server, a client would normally wait to validate the encrypted Finish message. Sending data before this handshake is completed means that the data exchanges could have taken place over an inconsistent connection (e.g., if a man-in-the-middle modified the handshake messages). This "attack" could compromise the TLS connection, although the client's encrypted information would be protected, since the client is still able to authenticate the server and use its cryptographic information [5].

4.3 Impact of page composition on web performance

Figure 4a shows the PLT for test web-pages plotted with respect to the number of objects using HTTP/1.1, HTTPS, and HTTP/2.

The PLT in our experiment was subject to very high variability (in some cases larger than 100%) from one trial to the next, which inevitably affected the uncertainty on the average PLT estimation even after a certain number of trials. This variability demonstrates the sensitivity of the PLT to network condition and the dynamics of the medium access control (in particular the dynamic bandwidth allocation as discusses further in the text).



(a) With PEP

(b) Without PEP (using network tunnel)

Figure 4: Page load time (PLT) with respect to number of objects for HTTP/1.x, HTTPS and HTTP/2. The plot links web page of the same size with lines. For each point on a line, the product of the number of object and their size is the same.

Despite the variability, the PLT shows for HTTP/1.1 and HTTPS a clear trend of increasing PLT with respect to the number of objects. The number of objects influenced the PLT more than the total page size. In contrast, for HTTP/2 the PLT is a function of the size of the page, but not the size of objects within the page (see plot (b)).

For example, downloading a web page consisting of a hundred objects of 5 KB (labelled *500 KB* in the figure) took around 35 s when using HTTPS, while a page of 15 100 KB objects (labelled *1500 KB*) too only 12 s.

This dependency of the PLT on the number of objects when using HTTP/1.1 is not surprising. A client can open only a certain number of connections towards the web server and request only one web object at a time per connection. Moreover, an HTTP application-layer PEP could prefetch large objects from the server and serialise them for a client connected via the satellite network. This avoids the time that an end-to-end TCP connection would spend opening its congestion window and reduces the dependency on object size (see plot (a)).

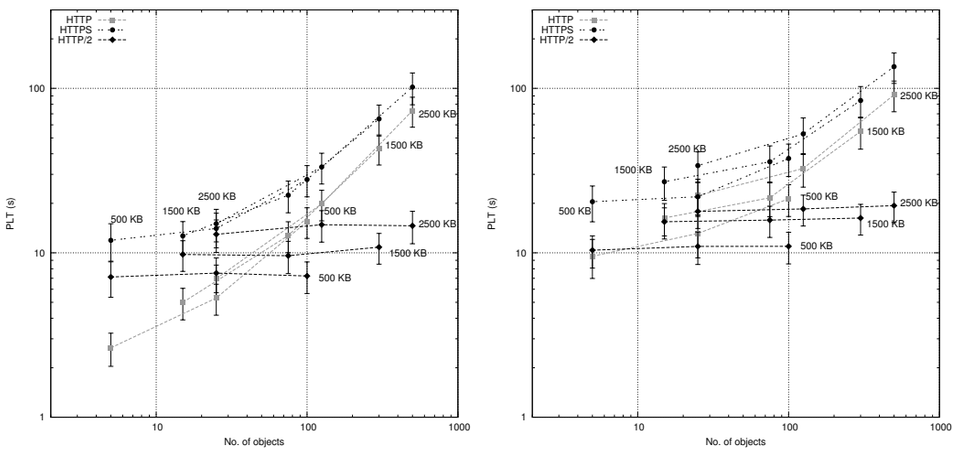
Figure 4a also shows that the PLT with HTTP/1.x was significantly lower than with HTTPS. This performance is limited because the HTTP/1.x connection is not persistent and represents the worst case for HTTPS. When connections are not persistent, a new TCP and TLS handshake needs to be performed for each object. The performance of HTTP/1.1 and HTTPS improves when persistence is enabled [5].

HTTP/2 (Fig 4a) resulted in a PLT of 10-12 seconds, which was not strongly dependent on the number of objects. Many more objects could be transmitted concurrently. The HTTP/2 performance also depended weakly on the page size. However, HTTP/2 flows were still unable to fully-utilise the capacity available on a high delay path and the transmission rate varies substantially (Figure 3). It was difficult to identify one specific factor limiting this performance, and the effect was attributed to interactions between the control-loop of the satellite network (bandwidth on demand mechanisms, queuing in network devices) and the HTTP/2 protocol (protocol design, default configuration and implementation). Section 4.6 describes how some of these problems could be mitigated by appropriate configuration of the web client.

4.4 Performance with a large end-to-end delay (no transport PEP)

The impact of using a PEP was evaluated in a set of tests that sent traffic using a network tunnel (using the point-to-point (PPP) protocol) between the client and the web server (Figure 1). This prevented the satellite PEP from intercepting transport headers or data, and hence disabled transport acceleration (the performance would be the same if a Virtual Private Network or IPsec had been used).

Figure 4b shows the HTTP performance without the mediation of a PEP. This performance exhibits more variability than when using a PEP, with an increasing trend of PLT with respect to the number of objects. Inspecting the traces showed that the large variations in RTT (of the order of seconds) was a root cause of the large performance variations. A likely reason for this variation was the delay introduced by the satellite return-link BoD scheme. Typical schemes seek to track variations in demand for capacity, removing capacity



(a) With PEP

(b) Without PEP (using network tunnel)

Figure 5: Page Load Time (PLT) with respect to the number of objects for HTTP, HTTPS and HTTP/2. The plot links web page of the same size with lines. For each point on a line, the product of the number of object and their size is the same.

allocations if a terminal has been idle for a period of time. This can cause a client to wait for a new capacity allocation after a server has been slow to respond to a request. In contrast, these delays can be avoided by a transport PEP that splits the connection (eliminating the capacity needed for transport protocol chatter and enabling cross-layer optimisation between the PEP and BoD methods). Disabling the PEP was found to increase delay variance, which also in turn impacted the ability for TCP to estimate an appropriate RTO, ultimately impacting performance.

The results show that the PLT for a 500 KB webpage using HTTP/2 ranged between 11 and 15 s. The results for HTTPS span a much larger interval, taking up to few minutes to download a page when there are more than one hundred objects. There was a clear performance improvement with HTTP/2, attributed to efficient multiplexing of the many small objects onto a single TCP connection. In these results, the transfer rate was not constrained by the peak SLA capacity.

Analysing the traces we found that several factors impacted the performance:

1. Performance over satellite was influenced by the connection retry timeout (*kMaxConnectRetryIntervalMs*) in the Chrome client configuration. This parameter set the time the browser waited after opening a connection before attempting to open a new connection (in Mozilla Firefox, the variable *network.http.connection-retry-timeout* has a similar purpose). The tested default setting was 250 ms, slightly larger than the default TCP initial RTO on several Unix-based systems (200 ms). When this parameter was less then the path RTT, it caused multiple connections to be opened and resulted in connection errors at the start.

2. The memory reserved by the client for the TCP connection receive buffer was about 128 KB. Although the TCP Window Scaling Option [32] was used (enabled in the majority of modern TCP/IP stacks) and the application was able to retrieve data quickly from the receive buffer, the TCP receiver advertised a window smaller than the bandwidth-delay product. The small advertised window is a result of limited buffer space reserved by this browser client.

3. The maximum number of concurrent active streams was limited to one hundred (the default value in an Apache modspdy server). This is a precautionary measure to constrain the amount of memory used by HTTP/2 streams. While the number was chosen to be sufficient for a terrestrial path, a path with a large bandwidth-delay product could easily utilise several hundred streams, if the object size were only a few kilobytes.

4. The browser had a timeout to reset the connection when idle for several seconds. In some of our tests, the HTTP session was reset more than once before completing download of the objects. At each reset, new TCP connections were created.

HTTP parameters in a web browser are typically configured to achieve adequate performance over common network paths. All these issues are related to design decisions

Table 1: PLT of HTTP/2 with IW=3 and IW=10.

page size (KB)	obj size (KB)	PEP		no PEP	
		IW3(s)	IW10(s)	IW3(s)	IW10(s)
500	5	10.8	9.5	26.7	23.0
	20	10.8	9.6	23.1	32.7
	100	10.2	11.3	24.8	23.2
1500	5	10.8	13.1	46.6	40.1
	20	9.7	12.1	52.5	34.8
	100	9.9	12.7	34.8	45.5
2500	5	12.1	15.6	68.4	49.9
	20	12.3	15.7	56.0	51.7
	100	11.9	15.0	60.4	64.9

that would be appropriate for a terrestrial Internet path. However, HTTP/2 offers mechanisms (such as SETTINGS and the UPDATE control frame) that allow configuration of the HTTP session based on the requirements of the client/server. Using these mechanisms it may be possible to optimise the parameters to improve performance of a session using a high bandwidth-delay path.

The next section illustrates how the performance could be increased by optimising the browser based on the network path being used.

4.5 Impact of the Initial Congestion Window

Table 1 shows the impact on PLT of increasing the IW at the web server from three to ten segments [26]. When a PEP was used, increasing IW at the server had limited impact, since a transport PEP already compensated for the effects of TCP startup. The larger IW was only used for the server-proxy loop without visible effect on the overall server-client connection (the latency may even increase due to limited buffer space). The larger IW may have benefited HTTP/2 performance if the client connected without the mediation of a PEP. However, in this case the performance advantage was overshadowed by the effects of RTT variation. Thus, the improvements from using an IW of 10 are difficult to quantify.

4.6 Performance over a path with Large Bandwidth-Delay Product

In our experiments, the small default connection-retry timeout caused Chrome to initiate three to five TCP connections at start-up before choosing one for data transfer and then dropping the unused ones. Connection drops also occurred during transmission when the path had been idle for a few seconds. In this case, however, these also had the side effect of causing the TLS session to reset and in some cases to fallback to using HTTP/1.1.

Figures 5a and 5b shows the PLT over satellite after Chrome was re-compiled with kMaxConnectRetryIntervalMs set to 1500 ms. (Mozilla Firefox makes this variable available from the configuration panel, in contrast to Chrome, which required the source code had to be changed.) The experiment was repeated with the PEP enabled (Figure 5a) and when PEP was disabled using a tunnel (Figure 5b).

Both set of results show a decrease in the PLT. This was principally due to the stability of the connection. With PEP enabled, HTTPS exhibited a lower PLT across the range of analysed object numbers, while the improvement for HTTP1/1.1 was more significant for transfers with a larger number of objects. With the PEP disabled, fine-tuning of the connection-retry timeout produced a substantial reduction in the PLT for all cases. In particular, HTTP/2 achieved the same performance as observed with a PEP. This suggests that a PEP may not be needed in presence of HTTP/2 traffic, provided a client had been appropriately configured.

5 Discussion

The results in the previous section show that tuning could significantly improve the performance of a web client over a satellite path. However, it is challenging to provide automatic calibration of web client parameters (e.g., a higher connection retry timeout) to accommodate the larger delay experienced on some network paths. To achieve suitable information for tuning requires knowledge of the network path characteristics. This information is not commonly provided by a network stack and simple measurements may be expected to result in mis-tuning. For example, a web client that relies exclusively on RTT measurements to initialise application-layer timeouts (assuming a network stack that exposes TCP's RTT samples or the RTO), could suffer significant delay if some of its connections are retarded

at startup. Delays at startup might occur because of a slow middle-box or an unsupported TCP option. In these cases timeouts are a quick way to escape this deadlock.

A web client configuration could be tuned by running an installation script on each user's computer, such as proxy auto-config (PAC) files or by proxy auto-configuration (such as web-proxy auto-discovery protocol - WPAD [33]). These methods, though, require manual intervention of the user. They can be expensive for an operator to deploy and maintain, and prone to errors (e.g., when clients connect to multiple infrastructures).

5.1 Implication and opportunities for PEPs when using long network paths

The limited performance of HTTP/1.x over long delay paths has resulted in deployment of PEPs [19]. These satisfy two needs - first, a desire to avoid individual users needing to install specific updates or changes in configuration because their network path happens to be long, and second to mitigate performance problems in the design and usage of HTTP/1.1.

PEPs are therefore currently considered an important tool by operators needing to support long network paths. They are widely used for satellite broadband access and also in mobile operator networks to ensure acceptable web performance without the need to update or configure client equipment [43].

One common transport PEP mechanism splits transport connections, allowing them to accelerate connection setup, and congestion control. Transparent application PEPs intercept web traffic, and assist cross-layer interaction with satellite MAC mechanisms. Deployed PEPs typically use a range of mechanisms and operate at multiple levels. Many are tailored to the networks in which they are deployed.

However, PEPs have many practical drawbacks:

- A Transport PEP by definition has to understand the syntax and protocol of the transport protocol that it is enhancing. Current PEPs hence only support a predefined set of transport protocols (e.g., may fail to offer benefit with the Stream Control Transport Protocol (SCTP), or packets encapsulated in other packets, such as SCTP/UDP or QUIC/UDP).

- A Transport PEP breaks the end-to-end semantics of the transport. This seemingly simple change has significant implications when the transport is updated or new options are introduced (such as a larger IW, improvements to loss recovery or congestion control or new options such as TCP Fast Open [36]). Lack of compatible support in a PEP results in options (or packets) being dropped preventing use of new extensions. (Even though enhancements such as TCP Fast Open are now widely deployed, our experiments showed that the PEP we used prevented the client using this option. It also negated the benefit of a larger IW).

- Some transport protocols by design do not reveal their protocol. Applications using a UDP-based transport (e.g., based on QUIC, Path Layer UDP Substrate, PLUS, or some other transport) may choose not to fix the protocol to allow more rapid evolution, or may employ end-to-end encryption to avoid middleboxes tampering with protocol exchanges. Such methods prevent use of traditional Transport or Application PEPs.

- Perhaps most significantly, current application PEPs are incompatible with the end-to-end security model of HTTPS and HTTP/2, and need to break this model to enable acceleration. The volume of encrypted HTTP traffic continues to grow [37] and many networks enable such proxies by installing root certificates in user devices to sign the HTTPS objects requested by the user [43]. This allows a TLS connection to be split at an application-layer PEP, but breaks the end-to-end integrity of the connection [38] and is only possible in scenarios where the network administrator has access to user devices (e.g., common for handheld devices on a mobile operator network [43]).

- All PEPs introduce an additional service element which can ossify the network, unless the element continues to be updated. e.g., tracking changes in application semantics, such as introduction of HTTP/2 methods, or changes in transport protocol design or usage.

In summary, the introduction of HTTP/2 and the new TCP optimisations demands changes to the way PEPs are used to avoid ossification of the network stack. Ossification is particularly undesirable for application protocols - where updates to protocols can be deployed over very short timescales (months rather than years) - much faster than PEPs are being updated. This could easily lead to a situation where an outdated PEP causes a performance problem.

One way to reduce the maintenance cost of deploying PEPs may be to separate the transport and application functions. Providing the transport can be updated either by maintaining an up-to-date Transport PEP or updating the user transport protocols, the Application PEP functions could be performed using a new type of HTTP/2 proxy.

PEPs could also be virtualised using methods such as the Network Function Virtualisation (NFV) framework introduced by ETSI [41]. NFV enables a software-defined networking (SDN) model where the network functions are maintained separately from the underlying hardware giving ISPs more flexibility in controlling and managing their networks. Network operators still need to continuously update network devices to benefit from HTTP/2 and new TCP optimisations, but may allow PEP functions to be more easily updated [40]. NFV solutions do not resolve the need for devices to intercept data - nor provide a clear direction for cross-layer optimisations (such as interacting with satellite BoD mechanisms).

5.2 Evolution to support longer network paths

In the previous section, we argued that deploying a PEP may result in ossification that conflicts with the desire to enable evolution, a core motivation in the design of HTTP/2. Since HTTP/2 has been designed to ease updates (e.g., implementations can ignore all unknown Frame types), we suggest further extensions are to be expected.

Our analysis of Application PEPs and split transport PEPs presented in Section 5 showed that HTTP/2 can be a viable alternative to deploying new PEP mechanisms. However, experience with the current implementation of this protocol indicates there are still small, but important, areas where performance variability is unlikely to be acceptable for complex web pages.

We therefore advocate an approach that reduces latency for long network paths by enabling auto-tuning within the HTTP and transport protocols to adapt the client and servers. This requires the protocols to understand the expected characteristics of the network path and its likely capacity and packet loss rate - allowing the client browser to set/request appropriate configuration parameters. Such information can be aggregated at the transport layer and could be made visible to applications through a new transport layer API [39]. We argue this approach will make the web more robust for a wide variety of paths, and can enable continued evolution.

A new approach to the design of transport stacks [39] could also enable browser clients to benefit from alternative transports - such as QUIC [11] or SCTP. SCTP using http 1.1 was evaluated in [35] and was shown to offer desirable features for high delay paths.

Reducing the reliance on network-specific enhancement of application and transport protocols by PEPs is expected to have additional benefits. An end-to-end approach seeks solutions that still offer benefit for content that cannot be proxied (e.g., dynamic web content, or when security mechanisms hide the payload data from middleboxes). Methods continue to work as new transport methods emerge, and even when tunnels and (opportunistic) transport encryption is used. Using standard methods means the methods much more likely to be compatible with commercial network accelerators designed for the general Internet (e.g., web content distribution networks, front-end load balancers and devices for traffic engineering).

The *push* mechanism, although not supported in current systems, has the potential to also improve latency if appropriately used in long-delay paths [3]. In a satellite context, this approach could eventually replace proprietary Application PEP functions (e.g., [42]).

There are proposals to introduce a new form of explicit web proxy, preserving the end-to-end security model, while permitting acceleration for certain content. A possibility discussed in the IETF httpbis working group is to define a method to signal the presence of intermediaries (middleboxes) that are able to cooperate with clients [34] to improve user web experience. This form of *explicit proxy* would be useful in a variety of scenarios [34], including an academic network, enterprises and Internet hot-spots. A major concern with the deployment of such a network device is a change to the security model, since this prevents end-to-end TLS integrity, and the new agents will be able to read the decrypted request and content. One potential solution could be to update the TLS protocol to enable multiple security contexts that explicitly permit a PEP (or other middlebox) to insert itself on the network path [38].

6 Conclusion

HTTP/2 was developed after substantial experience using HTTP/1.x, and has been designed to address the performance limitations of HTTP/1.x. It introduces better multiplexing, the use of persistent connections, content compression, and other mechanisms that directly reduce the latency experience by users. While the design of HTTP/2 was intended to be applicable to a wide range of network characteristics, the initial implementations have

naturally targeted common network conditions, where it has already been shown to offer significant reductions in page load time. This paper examined whether the design of a currently available implementation offers similar benefits when used over a long network path, such as that presented by a geostationary satellite network.

Using HTTP/1.1 over a satellite network that supported protocol acceleration, our experiments showed that HTTP/1.1 can outperform HTTP/2. However, significant performance improvements with HTTP/2 could have been achieved had the web client been configured appropriately for the satellite path delay. For example, although the default size of the TCP send and receive buffer did not have limit performance when the end-to-end connection was split by an intercepting transport PEP, when the PEP was removed this limited performance. Other parameters could also be tuned to achieve significant improvements in performance (e.g., the default initial connection size used in the Flow Control mechanism and the maximum number of permitted parallel streams).

When the page had a large number of objects, HTTP/2 outperformed HTTP(S). Although the exact benefit depends on the structure of the web page, a web page with more than a hundred HTTP/2 objects completed tens of seconds earlier.

Our experience with HTTP/2 has led us to conclude that an implementation of HTTP/2 can be tuned to improve performance over a long path, and may then form a credible alternative to using a PEP. However, we argue that such tuning needs to be automated, rather than requiring each client to be re-configured. Appropriate tuning and evolution of the transport and HTTP protocols is expected to overtake the ability of PEP mechanisms at the transport and application layers. By moving away from network-specific approaches, this also opens the possibility of benefiting from use of new transports and the continued development of new forms of HTTP/2 middleboxes.

References

[1] T. Berners-Lee, R. Fielding, H Frystyk, "Hypertext Transfer Protocol – HTTP/1.0", IETF Informational RFC 1945, May 1996.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol - HTTP/1.1", IETF Standard Track RFC 2616, Jun 1999.

[3] L. Caviglione et al., "A deep analysis on future web technologies and protocols over broadband GEO satellite networks" International Journal of Satellite Communications and Networking, 33(5) pp 451?472, Sep 2015.

[4] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control", IETF Standard Track RFC 5681, Sept 2009.

[5] B. Briscoe et al., "Reducing Internet Latency: A Survey of Techniques and their Merits", IEEE Communications Surveys and Tutorials, preprint, Nov 2014.

[6] J. Domenech, A. Pont, J. Sahuquillo, and J. Gil, "A user-focussed evaluation of web prefetching algorithms", ACM Computer Communications, 30(10), pp 2213-2224, 2007.

[7] S. Ramachandran, "Web metrics: Size and number of resources", http://code.google.com/speed/articles/web-metrics.html

[8] M. Belshe, R. Peon, "SPDY protocol - Draft 3.1", http://www.chromium.org/spdy/spdy-protocol/, 2016.

[9] M. Belshe, M. Peon, and M. Thomson, "Hypertext Transfer Protocol version 2", IETF Proposed Standard, RFC 7540, May 2015.

[10] HTTP/2 Adoption, [online] http://isthewebHTTP/2yet.com/measurements/adoption.html, daily updated.

[11] J. Roskind, "QUIC (Quick UDP Internet Connections), Multiplexed Stream Transport over UDP", Google technical report, [online], Apr 2012.

[12] R. Hamilton, J. Iyengar, I. Swett and A. Wilk, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2", IETF Work-in-Progress, Jan 2016.

[13] Iyengar, J. and I. Swett, "QUIC Loss Recovery and Congestion Control", IETF Work-in-Progress, Dec 2015.

[14] Y. Elkhatib, G. Tyson, and M. Welzl, "Can SPDY really make the web faster?", In proc. of IFIP Networking Conference, Tronhdeim, Jun 2014.

[15] Z. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How Speedy is SPDY?", proc. of 11th USENIX symposium on Network Systems Design and Implementation (NSDI'14), pp 387-393, Seattle, Apr 2014.

[16] L. Caviglione, and A. Gotta, "Characterizing SPDY over High Latency Satellite Channels", EAI Endorsed Transactions on Mobile Communications and Applications, 2(5), pp 1-10, Dec 2014.

[17] M. Luglio, C. Roseti, and F. Zampognaro, "SPDY multiplexing approach on long-latency links", in proc. of IEEE WCNC, pp 3450-3455, Instanbul, Apr 2014.

[18] M. Luglio, C. Roseti, and F. Zampognaro, "Resource optimization over DVB-RCS satellite links through the use of SPDY", WiOpt, Hammamet, May 2014.

[19] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations", IETF Informational RFC 3135, Jun 2001.

[20] R. Peon, and H. Ruellan, "HPACK: Header Compression for HTTP/2", IETF Standard RFC 7541, May 2015.

[21] B. Briscoe, HTTP/2 flow control <draft-ietf-httpbis-HTTP/2-17>, IETF Mailing List Archive https://lists.w3.org/Archives/Public/ietf-http-wg/2015JanMar/0525.html, Mar 2015.

[22] R. Braden, "T/TCP – TCP Extensions for Transactions Functional Specification", Historic RFC 1644, Jul 1994.

[23] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain, "TCP Fast Open", Experimental RFC 7413, Dec 2014.

[24] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window", IETF Proposed Standard, RFC 3390, Oct 2002.

[25] N. Dukkipati et al. "An Argument for increasing TCP's Initial Window", ACM SIGCOMM Computer Communication Review, 40(3), pp. 26-33, Jul 2010.

[26] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis, "Increasing TCP's Initial Window", Experimental RFC 6928, Apr 2013.

[27] Hughes Networks, "IP over Satellite (IPoS) - The Standard for Broadband over Satellite", 2007,[online: http://defense.hughes.com/resources/ip-over-satellite-ipos]

[28] Avanti satellite communications, http://www.avantiplc.com

[29] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", IETF Standard, RFC 5246, Aug 2008.

[30] J. Salowey and H. Zhou and P. Eronen and T. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", IETF Standard, RFC 5077, Jan 2008.

[31] A. Langley, N. Modadugu, and B. Moeller, "Transport Layer Security (TLS) False Start", draft-bmoeller-tls-falsestart-01, Internet Draft, Nov 2014.

[32] D. Borman, B. Braden,V. Jacobson, and R. Scheffenegger, "TCP Extensions for High Performance", IETF Proposed Standard, RFC 7323, Sep 2004.

[33] P. Gauthier, J. Cohen, M. Dunsmuir, and C. Perkins, "Web Proxy Auto-Discovery Protocol", Internet Draft, draft-ietf-wrec-wpad-01.txt, Work-in-Progress, Dec 1999.

[34] "IETF Hypertext Transfer Protocol (httpbis) Working Group", Transport Area, https://datatracker.ietf.org/wg/httpbis/

[35] P. Natarajan and R. Stewart, "Multistreamed Web Transport for Developing Regions", Proc. ACM SIGCOMM Workshop on Networked Systems for Developing Regions (NSDR 08), pp. 43-44, Aug 2008.

[36] "IETF TCP Maintenance and Minor Extensions WG Transfer Protocol (tcpm) Working Group", Transport Area, https://datatracker.ietf.org/wg/tcpm/

[37] P. Lepeska, "Trusted Proxy and the Cost of Bits", 90th IETF meeting, Toronto, https://www.ietf.org/proceedings/90/slides/slides-90-httpbis-6.pdf, Jul 2014.

[38] D. Naylor et al., "Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS", ACM SIGCOMM CCR, 45(4), pp. 199-212, Oct 2015.

[39] "A New Evolutive API and Transport-Layer Architecture for the Internet: The NEAT architecture", Deliverable 1.1, https://www.neat-project.org/wp-content/uploads/2016/02/D1.1.pdf

[40] Y. Ki and M. Chen, "Software-Defined Network Function Virtualization: A Survey", IEEE Access, vol 3., pp. 2542-2553, Dec 2015.

[41] ETSI GS NFV, "Network Functions Virtualisation (NFV); Architectural Framework", Oct 2013.

[42] Hughes TurboPage Web Accelerator, data sheet, http://business.hughes.com/resources/hughes-turbopage-web-accelerator, 2012.

[43] cite: https://www.cs.montana.edu/mwittie/publications/Goel16Detecting.pdf